Webology (ISSN: 1735-188X)
Volume 18, Number 6, 2021

Optimization Based Neural Network Classifiction Method For Software Defect Prediction

M. Subhashini[1] , A.Misbahulhuda[2]

[1]Assistant Professor in Department of Computer Science, Srimad Andavan Arts and Science College (Autonomous) (Affiliated to Bharathidasan University), Tiruchirappalli, Tamil Nadu, India.

[2]Research Scholar in Department of Computer Science, Srimad Andavan Arts and Science College (Autonomous) (Affiliated to Bharathidasan University), Tiruchirappalli, Tamil Nadu, India.

ABSTRACT
The software industry strives to enhance software quality by predicting bugs, removing bugs, and predicting fault-prone modules. Researchers have been drawn to this field because of its importance in the software industry. For software defect prediction, various strategies have been presented. Data mining using machine learning has been advocated as an important paradigm for software bug prediction in recent studies. The current level of software defect prediction has a number of flaws, including classification accuracy. Software defect databases, on the other hand, are unbalanced and known to be prone to errors due to their large size. This research offered a combined approach for software defect prediction and software bug prediction to address this issue. The suggested method combines Neural Networks with the FireFly Optimization technique. The FFO method optimises the weights of the Back Propagation Neural Network. The ideal minimises the BPNN classification error. The proposed FF-NN classification method is compared to existing classifiers such as Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Nave Bayes (NB) using a variety of evaluation metrics such as Accuracy, Sensitivity, Specificity, Precision, and error rates such as Miss Rate, False Positive Rate, and False Discovery Rate.

KEYWORDS: Software Defect, Classification, Prediction, Neural Network, Fire Fly Optimization.

1. INTRODUCTION

The need for software for diverse purposes has risen dramatically during the last two decades [1] [2]. To suit client demand, a large number of software applications for corporate or daily use are developed. Software quality is an unsolved issue as a result of mass production, resulting in inadequate performance for industrial and individual applications. To address this issue, software testing was developed, which aids in the discovery of faults or bugs in software applications and attempts to repair them [3] [4]. A great number of software applications are generated each year due to increased demand from current technology-based industries and business applications, yet software quality remains an ignored concern during this development. Software applications have become an indispensable component of daily life and business in recent years. A tiny software error in a corporate setting might result in a loss to the industry and impair customer satisfaction [5] [6]. Software testing-related concerns have become a major source of worry. As a result, an automated software testing procedure is necessary, which can increase performance and reduce installation costs. This problem can be avoided if the tester understands the reasons of potential faults and the overall software development process. This can help with not only lowering overall software development costs, but also better planning and execution of software development projects [7][8].

A defect or bug is defined by IEEE standards as a "inappropriate step, procedure, or data definition in a computer programme" [9]. In this article, the terms "software defects" and "bugs" are used interchangeably to refer to errors in software source code [10]. Users may experience undesirable system behaviour at any time as a result of system failure caused by defects or faults.

The software development life cycle is regarded as a fundamental contributor to software development in the software industry. Early defect prediction is becoming increasingly popular, and project managers are considering it a crucial demanding responsibility [11]. Complex issue domains, rising software application requirements, software performance uncertainty, and a complex development process are all recent developments in the software development sector. Despite comprehensive documentation and a well-organized procedure, some faults are unavoidable in the software development process, resulting in software performance decrease.

Various strategies for software defect minimization have been introduced in today's industry development. However, for proper software application analysis, these methodologies necessitate additional time, money, and resources. These efforts, on the other hand, can aid in the analysis of fault sources and the improvement of software performance. Software reliability is another term for software performance. According to software reliability, software applications are tested in an unpredictable environment and assessed to see if they are capable of operating in the environment for a specific period of time [12]. This method is based on software reliability probability estimation [13]. Software metrics are employed to complete this task, and it is discovered that a higher number of failure predictions necessitates more quality improvement resources, making it a difficult process. Several software defect prediction models based on software metrics have been proposed in recent decades. These models can aid in the early detection of problems and the development of trustworthy software.

Most defect prediction models created to date have used software metrics such as classical software metrics, object-oriented software metrics, and process metrics [14] [15]. Organizations utilise Pareto analysis for software quality measurement in real-time scenarios, where software metrics are combined with the greatest metric value for a specific software application. Although this technique improves performance, it is still unable to capture many errors, resulting in software testing performance decrease. According to the study, human review is ineffective at detecting faults in software modules. In a different scenario for software defect prediction, software measurements are used to create a statistical model for forecasting software problems. The regression or function-approximation problem analysis [16] is used to create these models. However, these methods do not produce effective results. This is due to the fact that each software's design is unique, with various function combinations, development teams, and third-party components. As a result, the software defect prediction produces an incorrect result. Furthermore, because of the variety of terms, no "critical value" for any of the software metrics can be determined, and parametric models such as linear models, Poisson regression, and quadratic models, among others, 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 Data Mining technique and computational intelligence for predicting the software defects.

3. ARTIFICIAL NEURAL NETWORK CLASSIFICATION

Artificial Neural Network (ANN) [17] [18] [19] [20] [21] [22] is a mathematical model that simulates neuron action in the human brain computationally by duplicating the brain's pattern and producing outputs depending on the learning of a set of training data. One of the most common neural network topologies is a multi-layer feedforward network with a back-propagation learning mechanism. It has been extensively researched and applied in a variety of fields. A neural network is typically made up of three layers: an input layer, an output layer, and an intermediate or hidden layer. The input vector is $\in R^n$ and $D = (X_1, X_2, .., X_n)^T$; the output of q neurons in the hidden layer are $Z = (Z_1, Z_2, .., Z_n)^T$; and the outputs of the output layer are $Y \in R^n, Y = (Y_1, Y_2, .., Y_n)^T$. Assuming that the weight and the threshold between the input layer and the hidden layer are w_{ij} and y_j, respectively, that the weight and the threshold between the hidden layer and output layer are w_{jk} and y_k respectively, the outputs of each neuron in a hidden layer and output layer are:

$$Z_j = f\left(\sum_{i=1}^{n} w_{ij} X_i - \theta_j\right)$$

$$Y_j = f\left(\sum_{j=1}^{q} w_{kj} Z_j - \theta_k\right)$$

where f() is a transfer function, which is the rule for transferring the neurons' summed input to their output, and is a way of adding non-linearity into the network design through a proper choice. The sigmoid function, which is monotonic growing and ranges from 0 to 1, is one of the most often utilised functions. A standard feed forward neural network with three (3) inputs, one (1) hidden layer with seven (7) neurons, and one (1) output layer was utilised to validate the performance of the proposed model.

4. FIREFLY OPTIMIZATION ALGORITHM

The firefly algorithm is a simulation of a firefly's natural activity in searching for mates and determining the best position based on its luminous qualities [23]. The programme assumes that firefly will be dispersed throughout space at random. Their brightness and appeal have a big influence on each other's attraction. The brightness of its own is determined by its position, and the direction of movement is determined by the brightness of its own. High brightness fireflies are constantly attracted to low luminance fireflies. Firefly will move randomly if the brightness is the same. Because the better the position, the higher the brightness. The firefly executes position iteration in the movement iteratively until it finds the ideal place, or the best solution to the function. The following formula explains the mathematical explanation of the optimization of the firefly algorithm:

The relative luminance formula of the firefly is:

$$I_{ij}(r_{ij}) = I_i e^{-\gamma r_{ij}^2}$$

Where I_{ij} represents the relative brightness between the firefly j and the firefly i; I_i represents the absolute brightness of the firefly i at r=0; r is light absorption coefficient, the algorithm believes that the light is weakened during the propagation process, resulting in a decrease in brightness and attractiveness as the relative distance of the firefly increases. r_{ij} represents the relative distance between the firefly j and the firefly i.

Attraction Strength formula:

$$\beta_{ij}(r_{ij}) = \beta_i e^{-r_{ij}^2}$$

β_0 is the attraction of fireflies for light sources and also the greatest attraction.

Firefly moving position formula:

$$\vec{x}_j(g+1) = \vec{x}_j(g) + \beta_{ij}(r_{ij})\left(\vec{x}_i(g) - \vec{x}_j(g)\right) + \alpha\vec{\varepsilon}_j$$

Where g is the number of iterations, $\vec{x}_j(g)$ and $\vec{x}_j(g)$ as the location of the firefly, α is the constant of the step factor on the [0,1], $\vec{\varepsilon}_j$ which is a random number vector of uniform distribution, Gaussian Distribution or other distribution.

Algorithm: Fire Fly Optimization

```
Start
    Define the objective function, $f(x), x = (x_i, \dots, x_d)T$
    Generate the initial population of fireflies $x_i (i = 1,2, \dots, n)$
    Determine the light intensity $l_i$ at $x_i$ from $f(x_i)$
    Determine the light absorption coefficient $\gamma$
        While $t <$ max generation
            Make a copy of population for movement function
        For i=1;n all n fireflies
            For $j = 1;$ i all n fireflies
                If $I_j > I_i$
                Move fireflies i and j in d- dimension;
                End if
                Attractiveness varies with distance r via $\exp^{[-\gamma r]}$
                Evaluate new solution and update light intensity
            End
        End
        Rank the fireflies and find the current best
    End
    Post process results and visualization
End
```

5. PROPOSED FIRE FLY BASED NEURAL NETWORK CLASSIFICATION

The simple Back Propagation Neural Network (BPNN) has three layer that includes an input layer, a hidden layer, and an output layer. Its back-propagation functioning performs the training and testing steps. The input data in every layer are adjusted by interconnection weight between the layers (w_{ij}), which demonstrates the relation of the ith node of the current layer to the jth node of the next layer. The key role of the hidden layer is to process the input layer information. The sum of total activation is assessed by a sigmoid transfer function. In this proposed FFNN classification method, the weights of the BPNN are optimized with FF algorithm.

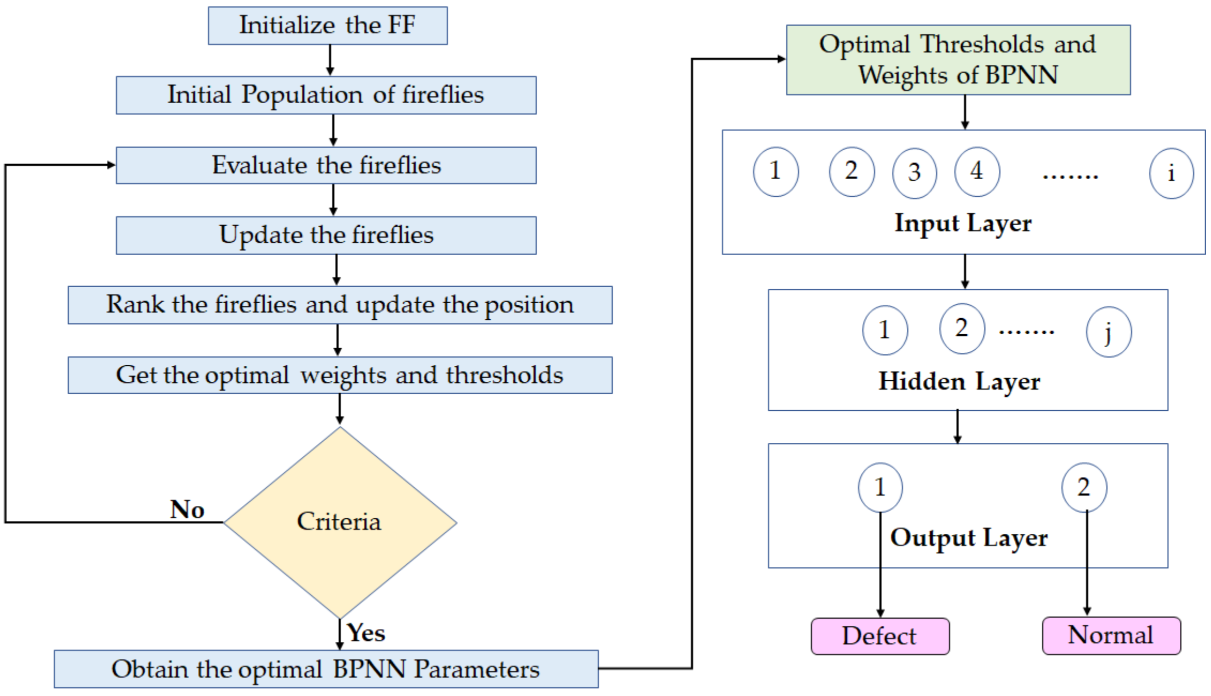
Initialize the FF
Initial Population of fireflies
Evaluate the fireflies
Update the fireflies
Rank the fireflies and update the position
Get the optimal weights and thresholds
Criteria
No
Yes
Obtain the optimal BPNN Parameters
Optimal Thresholds and Weights of BPNN
1 2 3 4 i
Input Layer
1 2 .... j
Hidden Layer
1 2
Output Layer
Defect
Normal

Figure 1: Flowchart of the Proposed Fire Fly based Neural Network Classification Method

```
Algorithm 2: Proposed FFNN Classification Method
Allocate all the inputs and on output
Initialize weights between -1 and 1.
Repeat
    For every input in the training set
        For each layer in the network
            For each node in the layer.
                Determine the objective Function by FF.
                Generate the initial population of fireflies $x_i (i = 1,2, \dots, n)$
                Determine the light intensity $l_i$ at $x_i$ from $f(x_i)$
```

```
                    Determine the light absorption coefficient $\gamma$
                    Calculate the summation of the inputs weight + threshold
                    Calculate the activation function.
                End
            End
        For every the output node
            Calculate the error function.
        End
        For all hidden layer
            For every node in the layer
                Calculate the Error function.
                Update the weights in the network with using Firefly moving position formula.
                Evaluate the new solution and update light intensity.
            End
        End
        Calculate the Error Function
        End
While ((iteration < maximum iterations) & (Error Function is > Criteria))
```

6. RESULT AND DISCUSSION

6.1 Description of the Dataset

The software defect dataset (jm1) is considered from the Kaggle Repository [24]. The following features are in the software defect dataset is given in the table 1.

Table 1: Dataset Description of the Software Defect Prediction

Sl.No	Feature Name	Feature Description
1	Loc	numeric % McCabe's line count of code
2	v(g)	numeric % McCabe "cyclomatic complexity"
3	ev(g)	numeric % McCabe "essential complexity"
4	iv(g)	numeric % McCabe "design complexity"
5	n	numeric % Halstead total operators + operands
6	v	numeric % Halstead "volume"
7	l	numeric % Halstead "program length"
8	d	Numeric % Halstead "difficulty"
9	i	numeric % Halstead "intelligence"
10	e	numeric % Halstead "effort"
11	b	numeric % Halstead
12	t	numeric % Halstead's time estimator
13	lO Code	numeric % Halstead's line count
14	lO Comment	numeric % Halstead's count of lines of comments
15	lO Blank	numeric % Halstead's count of blank lines

16	lO Code And Comment	Numeric
17	uniq_Op	numeric % unique operators
18	uniq_Opnd	numeric % unique operands
19	total_Op	numeric % total operators
20	total_Opnd	numeric % of the flow graph
21	defects	False: The module has no defects True: The module has one or more defects

6.2 Performance Metrics

Table 2 depicts the performance metrics used in this research paper to evaluate the proposed EOBFS method.

Table 2: Performance Metrics used in this research paper

Metrics	Equation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$
Sensitivity	$\frac{TP}{TP + FN}$
False Positive Rate (FPR)	$\frac{FP}{TN + FP}$
Precision	$\frac{TP}{TP + FP}$
Specificity	1- False Positive Rate (FPR)
Miss Rate	1-Sensitivity
False Discovery Rate	1- Precision

6.3 Performance Analysis of the Proposed FFNN classification Method

The performance of the proposed FFNN classification method is evaluated with existing other classifiers like Support Vector Machine (SVM), K-Nearest Neighbor (KNN) and Naïve Bayes (NB) with the original dataset and other feature selection methods like Proposed Enhanced Optimization based Feature Selection (EOBFS), Chi-Square (CS), Seagull Optimization Algorithm (SOA), Information Gain (IG), and Particle Swarm Optimization (PSO) processed datasets.

Table 3 depicts the Classification Accuracy (in %) obtained by the Proposed FFNN, SVM, KNN and NB using Original Dataset, Proposed EOBFS, CS, SOA, IG and PSO feature selection methods processed datasets. From the table 3, it is shown that the proposed FFNN classification with proposed EOBFS method gives better accuracy than the other classifiers using feature selection methods.

Table 3: Classification Accuracy (in %) obtained by Proposed FFNN, SVM, KNN and NB classification method using original dataset, Feature Selection processed datasets

Feature Selection Methods	Classification Accuracy (in %) by Classification Techniques			
	Proposed FNN	SVM	KNN	NB
Original dataset	58.74	54.47	46.72	44.44
Proposed EOBFS Method	96.85	92.86	83.38	79.74
CS	79.26	74.94	64.92	59.53
SOA	80.41	75.78	72.95	69.13
IG	68.32	65.65	62.84	58.63
PSO	66.55	63.79	59.74	56.64

Table 4 depicts the Sensitivity (in %) obtained by the Proposed FFNN, SVM, KNN and NB using Original Dataset, Proposed EOBFS, CS, SOA, IG and PSO feature selection methods processed datasets. From the table 4, it is shown that the proposed FFNN classification with proposed EOBFS method gives increased Sensitivity than the other classifiers using feature selection methods.

Table 4: Sensitivity (in %) obtained by Proposed FFNN, SVM, KNN and NB classification method using original dataset, Feature Selection processed datasets

Feature Selection Methods	Sensitivity (in %) by Classification Techniques			
	Proposed FFNN	SVM	KNN	NB
Original dataset	59.63	55.58	46.63	43.33
Proposed EOBFS Method	96.47	93.68	81.49	78.81
CS	77.24	73.92	69.84	67.24
SOA	79.68	75.87	65.83	60.64
IG	67.14	62.97	61.95	59.72
PSO	65.22	59.86	58.65	55.42

Table 5 depicts the False Positive Rate (in %) obtained by the Proposed FFNN, SVM, KNN and NB using Original Dataset, Proposed EOBFS, CS, SOA, IG and PSO feature selection methods processed datasets. From the table 5, it is shown that the proposed FFNN classification with proposed EOBFS method gives reduced FPR than the other classifiers using feature selection methods.

Table 5: False Positive Rate (in %) obtained by Proposed FFNN, SVM, KNN and NB classification method using original dataset, Feature Selection processed datasets

Feature Selection Methods	False Positive Rate (in %) by Classification Techniques			
	Proposed FFNN	SVM	KNN	NB
Original dataset	48.14	52.72	56.29	58.78
Proposed EOBFS Method	4.52	9.83	12.32	15.63
CS	14.39	20.53	29.29	31.56

SOA	18.96	26.62	30.71	33.38
IG	31.41	37.92	43.62	46.93
PSO	36.74	40.51	45.44	47.82

Table 6 depicts the Precision (in %) obtained by the Proposed FFNN, SVM, KNN and NB using Original Dataset, Proposed EOBFS, CS, SOA, IG and PSO feature selection methods processed datasets. From the table 6, it is shown that the proposed FFNN classification with proposed EOBFS method gives increased Precision than the other classifiers using feature selection methods.

Table 6: Precision (in %) obtained by Proposed FFNN, SVM, KNN and NB classification method using original dataset, Feature Selection processed datasets

Feature Selection Methods	Precision (in %) by Classification Techniques			
	Proposed FFNN	**SVM**	**KNN**	**NB**
Original dataset	63.74	57.92	54.81	51.85
Proposed EOBFS Method	95.41	91.43	82.62	78.79
CS	80.28	75.36	72.29	69.83
SOA	81.27	75.63	70.71	66.92
IG	69.35	64.92	61.83	59.88
PSO	65.42	61.31	59.62	56.74

Table 7 depicts the Specificity (in %) obtained by the Proposed FFNN, SVM, KNN and NB using Original Dataset, Proposed EOBFS, CS, SOA, IG and PSO feature selection methods processed datasets. From the table 7, it is shown that the proposed FFNN classification with proposed EOBFS method gives increased Specificity than the other classifiers using feature selection methods.

Table 7: Specificity (in %) obtained by Proposed FFNN, SVM, KNN and NB classification method using original dataset, Feature Selection processed datasets

Feature Selection Methods	Specificity (in %) by Classification Techniques			
	Proposed FFNN	**SVM**	**KNN**	**NB**
Original dataset	51.86	47.28	43.71	41.22
Proposed EOBFS Method	95.48	90.17	87.68	84.37
CS	85.61	79.47	70.71	68.44
SOA	81.04	73.38	69.29	66.62
IG	68.59	62.08	56.38	53.07
PSO	63.26	59.49	54.56	52.18

Table 8 depicts the Miss Rate (in %) obtained by the Proposed FFNN, SVM, KNN and NB using Original Dataset, Proposed EOBFS, CS, SOA, IG and PSO feature selection methods processed datasets. From the table 8, it is shown that the proposed FFNN classification with proposed EOBFS method gives reduced miss rate than the other classifiers using feature selection methods.

Table 8: Miss Rate (in %) obtained by Proposed FFNN, SVM, KNN and NB classification method using original dataset, Feature Selection processed datasets

Feature Selection Methods	Miss Rate (in %) by Classification Techniques			
	Proposed FFNN	SVM	KNN	NB
Original dataset	40.37	44.42	53.37	56.67
Proposed EOBFS Method	3.53	6.32	18.51	21.19
CS	22.76	26.08	30.16	32.76
SOA	20.32	24.13	34.17	39.36
IG	32.86	37.03	38.05	40.28
PSO	34.78	40.14	41.35	44.58

Table 9 depicts the False Discovery Rate (in %) obtained by the Proposed FFNN, SVM, KNN and NB using Original Dataset, Proposed EOBFS, CS, SOA, IG and PSO feature selection methods processed datasets. From the table 9, it is shown that the proposed FFNN classification with proposed EOBFS method gives reduced FDR than the other classifiers using feature selection methods.

Table 9: False Discovery Rate (in %) obtained by Proposed FFNN, SVM, KNN and NB classification method using original dataset, Feature Selection processed datasets

Feature Selection Methods	False Discovery Rate (in %) by Classification Techniques			
	Proposed FFNN	SVM	KNN	NB
Original dataset	36.26	42.08	45.19	48.15
Proposed EOBFS Method	4.59	8.57	17.38	21.21
CS	19.72	24.64	27.71	30.17
SOA	18.73	24.37	29.29	33.08
IG	30.65	35.08	38.17	40.12
PSO	34.58	38.69	40.38	43.26

7. CONCLUSION

One of the most important components of software is its quality. Software designs are becoming more sophisticated as demand grows, increasing the likelihood of software failures. Testers correct bugs in software to increase its quality. As a result, defect analysis increases software quality dramatically. The increased complexity of software also means a higher number of flaws, making manual detection a time-consuming operation. In this paper, enhancement of the Back Propagation Neural Network is done using FireFly Optimization (FFO) algorithm. The weights and threshold of the BPNN is optimized using FF algorithm. From the results obtained for the Proposed FFNN based classification method for the classification and prediction of software defect, the proposed FFNN based classification performs better in terms of Accuracy, Sensitivity, Specificity, Precision and it also reduced the error rates like FPR, Miss rate and FDR with feature selection processed datasets than other classifiers like SVM, KNN and NB .

REFERENCES

[1] Li, Zhiqiang, Xiao-Yuan Jing, and Xiaoke Zhu. "Progress on approaches to software defect prediction." Iet Software 12.3 (2018): 161-175.

[2] Rathore, Santosh S., and Sandeep Kumar. "A study on software fault prediction techniques." Artificial Intelligence Review 51.2 (2019): 255-327.

[3] Akmel, Feidu, Ermiyas Birihanu, and Bahir Siraj. "A literature review study of software defect prediction using machine learning techniques." Int. J. Emerg. Res. Manag. Technol 6.6 (2017): 300-306.

[4] Singh, Praman Deep, and Anuradha Chug. "Software defect prediction analysis using machine learning algorithms." 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence. IEEE, 2017.

[5] Malhotra, Ruchika. "A systematic review of machine learning techniques for software fault prediction." Applied Soft Computing 27 (2015): 504-518.

[6] Catal, Cagatay. "Software fault prediction: A literature review and current trends." Expert systems with applications 38.4 (2011): 4626-4636.

[7] Catal, Cagatay, and Banu Diri. "A systematic review of software fault prediction studies." Expert systems with applications 36.4 (2009): 7346-7354.

[8] Vandecruys, Olivier, et al. "Mining software repositories for comprehensible software fault prediction models." Journal of Systems and software 81.5 (2008): 823-839.

[9] Prasad, M. C., Lilly Florence, and Arti Arya. "A study on software metrics based software defect prediction using data mining and machine learning techniques." International Journal of Database Theory and Application 8.3 (2015): 179-190.

[10] Chen, Yuan, et al. "Research on software defect prediction based on data mining." 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE). Vol. 1. IEEE, 2010.

[11] Malhotra, Ruchika. "A systematic review of machine learning techniques for software fault prediction." Applied Soft Computing 27 (2015): 504-518.

[12] Singh, Pradeep, and Shrish Verma. "An efficient software fault prediction model using cluster-based classification." Int. J. Appl. Inf. Syst 7.3 (2014): 35-41.

[13] Punitha, K., and S. Chitra. "Software defect prediction using software metrics-A survey." 2013 International Conference on Information Communication and Embedded Systems (ICICES). IEEE, 2013.

[14] Azeem, Naheed, and Shazia Usmani. "Analysis of data mining based software defect prediction techniques." Global Journal of Computer Science and Technology (2011).

[15] Jacob, Shomona Gracia, and Geetha Raju. "Software defect prediction in large space systems through hybrid feature selection and classification." Int. Arab J. Inf. Technol. 14.2 (2017): 208-214.

[16] Thota, Mahesh Kumar, Francis H. Shajin, and P. Rajesh. "Survey on software defect prediction techniques." International Journal of Applied Science and Engineering 17.4 (2020): 331-344.

[17] Subhashini, M., & Gopinath, R., Employee Attrition Prediction in Industry using Machine Learning Techniques, International Journal of Advanced Research in Engineering and Technology, 11(12), 3329-3341 (2020).

[18] Rethinavalli, S., & Gopinath, R., Classification Approach based Sybil Node Detection in Mobile Ad Hoc Networks, International Journal of Advanced Research in Engineering and Technology, 11(12), 3348-3356 (2020).

[19] Poornappriya, T.S., & Gopinath, R., Application of Machine Learning Techniques for Improving Learning Disabilities, International Journal of Electrical Engineering and Technology (IJEET), 11(10), 403-411(2020).

[20] Poornappriya, T.S., & Gopinath, R., Employee Attrition in Human Resource Using Machine Learning Techniques, Webology, 18(6), 2844-2856 (2021).

[21] Priyadharshini, D., Gopinath, R., Poornappriya, T.S., A fuzzy MCDM approach for measuring the business impact of employee selection, International Journal of Management, 11(7), .1769-1775 (2020).

[22] Poornappriya, T.S., & Gopinath, R., Plant Disease Identification using Artificial Intelligence Approaches, International Journal of Electrical Engineering and Technology (IJEET), 11(10), 392-402 (2020).

[23] Kaveh, A., R. Mahdipour Moghanni, and S. M. Javadi. "Optimum design of large steel skeletal structures using chaotic firefly optimization algorithm based on the Gaussian map." Structural and Multidisciplinary Optimization 60.3 (2019): 879-894.

[24] https://www.kaggle.com/datasets/semustafacevik/software-defect-prediction?resource=download&select=jm1.csv