Received: April 5, 2018

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Software Fault Prediction based on GSO-GA Optimization with Kernel based SVM Classification

Yogomaya Mohapatra[1]* **Mitrabinda Ray[2]**

[1]*Orissa Engineering College, Bhubaneswar, Odisha, India*
[2]*Siksha 'O' Anusandhan University, Bhubaneswar, Odisha, India*
* Corresponding author's Email: mohapatrayogomaya@gmail.com

Abstract: The two main challenges of Software Defect Prediction are high dimensionality and class imbalance. The first challenge is high dimensionality where the number of features extracted from software modules becomes larger due to growth in size and complexity of modern software systems. The extracted features may be redundant or irrelevant. The problem of high dimensionality can be solved by an important pre-processing procedure that is feature selection. The second challenge is class-imbalanced data, where the major defects in a software system are found in few modules. The earlier methods failed to provide good solution for class imbalance and high dimensionality. Also, the prediction accuracy of the existing methods is low. To overcome these major drawbacks an optimal Group Search Optimization –Genetic Algorithm (GSO-GA) based fault prediction in software testing with kernel based SVM classification is used for improving the software quality. To evaluate the performance of the proposed approach that is hybrid Group Search Optimization-Genetic Algorithm (GSO-GA) based fault prediction is compared with existing Group Search Optimization (GSO). The GSO-GA based models have produced better results and accuracy in terms of existing software metrics like Average Percentage of Faults Detected (APFD), Problem Tracking Reports (PTR), and time and memory usage. From the experimental results, we observe that the average percentage of faults detected in our proposed approach is higher than the existing method.

Keywords: Software fault prediction, Prediction, Group search optimization, Genetic algorithm.

1. Introduction

Software development organizations are under more pressure than any time in recent times. Development costs keep on rising. Minimizing defects is an efficient approach to hold development costs down, which is a main concern for any association [1]. A huge amount of research in Software Engineering has been devoted to improve the efficiency of testing. Among these, considerable efforts have been aimed towards the definition of techniques able to predict the components of a software system that more likely will contain faults [2]. Regression testing provides many benefits such as enabling refactoring of code with high confidence to catching product faults or functional regressions prior to product release [3]. In software engineering, defect prediction can precisely estimate the most defect-prone software components and help software engineers allocate limited resources to those bits of the systems that are most likely to contain defects in testing and maintenance phases [4]. Software fault prediction approaches use previous software metrics and fault data to predict fault-prone modules for the next release of software [5].

Researchers have identified a bunch of problems in this area of software defect prediction and have tried to offer the answers as well [6]. As recent software system has grown in size and complexity, quality assurance such as testing and inspection have become increasingly important not only for software developers but also for software purchases who are responsible for acceptance testing and/or software services deployment [7]. Software quality models generally predict, for a program module, either the number of defects it is likely to have or

the quality-based risk category it belongs to, e.g., faultprone (*fp*) or not-fault-prone (*nfp*) [8].

The number of features to be extracted increases as the size of the software increases and many of these features may be redundant or irrelevant [9].

To overcome the major drawbacks of high dimensionality and class imbalance, to reduce the error rate we have used software fault prediction based on GSO-GA optimization with kernel based SVM classification in this paper. The contribution of this paper is to explore the capabilities of (GSO-GA) for software fault prediction. We build our fault prediction model using (GSO-GA) and kernel based SVM in JAVA platform. The performance of the constructed model is evaluated in terms of software metrics like APFD, PTR, execution time and memory usage.

The rest of the paper is organized as follows. Section 2 contains related work. Section 3 explains the proposed methodology- (GSO-GA) based fault prediction model. Section 4 gives the experimental results with comparative analysis. Finally, the paper is concluded in Section 5.

2. Related work

Software reliability prediction plays an important role in the analysis of software quality and balance of software cost. Recently Support Vector Regression (SVR) has been widely applied to solve nonlinear predicting problems and has obtained good performance in many situations, but it is difficult to optimize SVR's parameters.

C. Jin et al. [10] have proposed Estimation of Distribution Algorithms (EDA) to maintain diversity of population and hybrid Improved Estimation Distribution Algorithms (IEDA) and SVR model, called IEDA-SVR to optimize parameters of SVR and predict software reliability.

R. W. Li, et al. [11] has proposed a three-way decisions framework for cost-sensitive software defect prediction. Experimental results on NASA data sets show that the two-stage classification method which integrates three-way decisions and ensemble learning can obtain higher accuracy and a lower decision cost in comparison to the traditional two-way decision method.

X. Yang et al. [12] have introduced a learning-to-rank approach software defect prediction model by optimizing the ranking performance.

L. Yu, et al. [13] have proposed a novel evolutionary programming (EP) based asymmetric weighted least squares support vector machine (LSSVM) ensemble learning methodology.

H. Mahalingap, *et al.* [14], has proposed a paper, a new classification and prediction methodology is put forth to progress the accuracy of defect forecast based on Cost Random Forest algorithm (CRF) which reduces the effects of faults in irrelevant software modules. The proposed algorithm predicts the quantity of faults present in the modules of software in less time and classify based on measures of similarity obtained from Robust Similarity clustering technique.

Prediction of fault proneness of modules in software is one of the ways to ensure the achievement of software quality and reliability. N. S. Agrawa *et al.* [15], has proposed few models for detecting software fault prone modules, the intend of our work is to increase the reliability of the software by using an approach named Rough Fuzzy c-means (RFCM) clustering algorithm to analyze the fault proneness of the software modules under test.

3. Proposed methodology

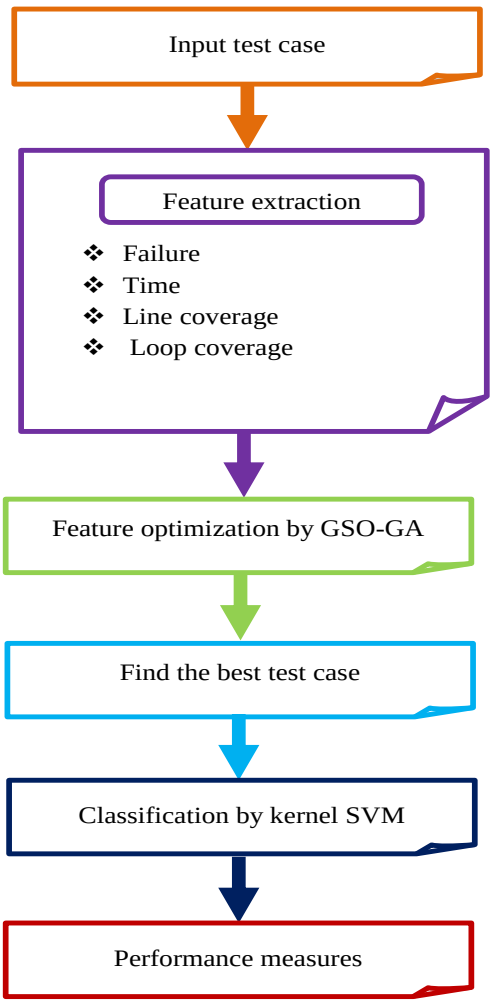


Figure.1 Proposed kernel SVM based software defect prediction

The proposed optimal Group Search Optimization – Genetic Algorithm (GSO-GA) fault prediction model is divided into 4 phases as shown in the Fig. 1.

The 4 phases are feature extraction, feature optimization, finds the best test case, classification by kernel SVM. Initially the test cases are taken as input for our study. The input test cases are given to the feature extraction phases. In this phase, we are extracting the features like failure, time, line coverage and loop coverage. Then the extracted test cases are optimized with the aid of GSO-GA algorithm. In this phase, we are mainly focused on feature optimization. These optimized features are used to find the best test cases. These best test cases are classified with the aid of kernel based SVM to predict the defects. Finally, performance analysis is performed. The proposed work is implemented in JAVA platform. The proposed work is compared with the existing methods and algorithms to prove that the proposed work is the best one.

3.1 Feature extraction

Initially the test cases are taken as input for our study. Input test cases are given to the feature extraction phase. In this phase we are extracting the features like failure, time, line coverage, and loop coverage. These features are taken from the test cases.

3.1.1. Line coverage

Line coverage is as well recognized as the statement coverage or segment coverage. Only correct circumstances are enfolded using line coverage. It as well events the excellence of the code and makes sure the flow of dissimilar path in that code.

$$Line\ Coverage = \frac{No.of\ Line\ excercised}{Total\ No.of\ Lines} \quad (1)$$

3.1.2. Loop coverage

These coverage metrics hear says whether each loop body is applied zero times, exactly once and more than once. This metrics reports whether loop body is applied exactly once and more than once for do-while loops. And also, while-loops and for-loops achieve more than once. This information is not accounted using other coverage metrics. Similarly, all the test cases are produced. Next resulting test case generation is optimized in to the GSO-GA algorithm. Since we will produce test cases in each time it encloses some resemblance on each time.

3.2 Feature optimization

The extracted test cases are optimized with the aid of GSO-GA algorithm. In this phase we are mainly focused on feature optimization. We also considered the problems of existing methods and addressed such limitations with the aid of our proposed optimal GSO-GA algorithm

3.2.1. Group Search Optimization-Genetic Algorithm (GSO-GA)

In this algorithm, the population is termed as a group and the individuals residing within the group are known as members. The members within a group are of three kinds, namely, the producers, the scroungers and the rangers. The activity of the producers as well as the scroungers relies on the PS model. The rangers move in an arbitrary manner.

Producers: These members go in search of resources.

Scroungers: These members link the resources, which the producer discovers.

Rangers: These are the members that make movements in an arbitrary manner and perform searching in an organized way, so that efficient finding of resources could be achieved.

Φ_{max} = maximum search angle	d_{max} = maximum search distance
F_i = Fitness computation	τ_{max} = maximum turning angle
d_{Ui} = upper limit	d_{Li} = lower limit
n = dimension of search space	Ψ = head angle
Z_i = producer	ε = uniform random sequence
L_i = members direction	

Initialize the search solution as well as the head angle:

The solution that is obtained after searching takes the thickness, period, the wall size and the temperature applied into account. Z_i represents producer.

$$Z_i = \begin{matrix} Z_{11} & Z_{12} & .... & Z_{1m} \\ Z_{21} & Z_{22} & .... & Z_{2m} \\ Z_{n1} & Z_{n2} & .... & Z_{nm} \end{matrix} \quad (2)$$

For every individual, the head angle can be stated as in Eq. (3).

$$\Psi_i^s = \left(\Psi_{i1.....}^s \Psi_{i(n-1)}^s\right) \quad (3)$$

The member's direction of search relies on the head angle and it is given as in Eq. (4).

$$L_i^s(\Psi_i^s) = \left(l_{i1\ldots\ldots l_{i(n)}}^s\right) \quad (4)$$

Polar and Cartesian coordinate transformation is employed to assess the direction of search in accordance with the head angle.

$$L_{i1}^s = \Pi_{p=1}^{n-1} \cos\left(\Psi_{ip}^s\right) \quad (5)$$

$$L_{ij}^s = \sin\left(\Psi_{i(j-1)}^s \Pi_{p=j}^{n-1} \cos\left(\Psi_{ip}^s\right)\right) \quad (6)$$

Where $(j = 2 \ldots n-1)$

$$L_{in}^s = \sin\left(\Psi_{i(n-1)}^s\right) \quad (7)$$

Fitness function

Optimization in the mathematical modeling can be accomplished using the sigmoid function, when it is included in the process involving artificial neural network. The optimized mathematical model can be yielded with the optimization of α_{ij} and β_{ij} in the function. The value of error in the mathematical model can be computed as the difference between the original value and the obtained value.

$$f_i = \sum_{j=1}^{h} \alpha_j \times \left[\frac{1}{1+exp\left(-\sum_{i=1}^{N} Z_i \beta_{ij}\right)}\right] \quad (8)$$

$$F_i = Original - Obtained \quad (9)$$

Find the producer Z_p of the group:

The member with the best fitness of Z_i is called as the producer and it is specified as Z_p.

Producer performance

During the execution of the GSO algorithm, the activity of the producer Z_p at 's' iteration can be elucidated as follows:

(i) The producer performs the scanning operation at zero degree

$$Z_z = Z_p^s + \varepsilon_1 d_{max} L_p^s(\Psi^s) \quad (10)$$

(ii) The producer performs the scanning operation at the right hand side hypercube

$$Z_r = Z_p^s + \varepsilon_1 d_{max} L_p^s\left(\Psi^s + \varepsilon_2 {}^{\Phi_{max}}/_2\right) \quad (11)$$

(iii) The producer performs the scanning operation at the left hand side hypercube

$$Z_l = Z_p^s + \varepsilon_1 d_{max} L_p^s\left(\Psi^s - \varepsilon_2 {}^{\Phi_{max}}/_2\right) \quad (12)$$

Where, ε_1 points to a normally distributed random number with zero mean and unity standard deviation; ε_2 stands for a uniformly distributed random sequence that takes value between zero and one. The maximum search angle Φ_{max} can be expressed as

$$\Phi_{max} = \frac{\pi}{c^2} \quad (13)$$

Here, the constant c can be stated as:

$$C = round\left(\sqrt{n+1}\right) \quad (14)$$

Where, n denotes the dimension of the search space.

$$\therefore \Phi_{max} = \frac{\pi}{n+1} \quad (15)$$

The computation of maximum search distance d_{max} involves the following equations.

$$\mathrm{d}_{max} = \|d_{U-}d_L\| = \sqrt{\sum_{i=1}^{n}(d_{Ui} - d_{Li})^2} \quad (16)$$

Where, d_{Ui} and d_{Li} indicates the upper limit and the lower limit of i^{th} dimension, respectively. The best location containing the most useful resource can be obtained with the help of Eqs. (16), (17), and (18). The present best location would take a new best location, if its resource is found as not better than that in the new location. Else, the producer will maintain its location and turn its head according to the head angle direction that is arbitrarily generated using Eq. (19).

$$\Psi^{s+1} = \Psi^s + \varepsilon_2 \tau_{max} \quad (17)$$

Where, τ_{max} indicates the maximum turning angle that is computed using the following equation.

$$\tau_{max} = \frac{\Phi_{max}}{2} \quad (18)$$

When the producer is unable to identify a better position even after the completion of m iterations, its head would then assume its initial position as given in Eq. (22).

$$\Psi^{s+c} = \Psi^s \quad (19)$$

Scrounger performance

In all iterations, several members that exclude the producer are also chosen and they are called as scroungers. The scrounging action of GSO usually involves the area copying activity. During the s^{th} iteration, the activity of area copying that the i^{th} scrounger performs can be modeled as a motion to reach the producer in a closer way and it is expressed as:

$$Z^{s+1} = Z_i^s + \varepsilon_3 \, O\left(Z_p^s - Z_i^s\right) \quad (20)$$

Where, O specifies the Hadamard product that computes the product of the two vectors in an entry-wise manner and ε_3 denotes a uniform random sequence lying in the interval of (0, 1). The i^{th} scrounger continues its searching activity to make a choice of the better occasion for linking. The designing of the scrounging action involves the turning of the head in the i^{th} scrounger to a novel and arbitrarily generated angle.

Ranger performance

The rangers are the remaining members of the group, which have been displaced from their present location. The rangers can also find the resources effectively through random walks or an organized searching procedure.

$$d_i = c.\varepsilon_1.d_{max} \quad (21)$$

The random walk to a novel point can be expressed as:

$$Z^{s+1} = Z_i^s + d_i L_i^s(\Psi^{s+1}) \quad (22)$$

Once the entire process gets completed, the fitness of the updated solution is evaluated. The best solution will be gained, if the process is repeated for 's' number of iterations. The Group Search Optimization algorithm (GSO) and the related mathematical modeling allow the temperature of the wall to be envisaged effectively. Here, we include genetic algorithm for GSO weight updation it will be clearly explained on below section.

3.2.2. Genetic algorithm (GA)

The novel adaptive genetic algorithm is used to optimize the weights. In case A possesses fitness superior to that of B, then A is chosen, ignoring B. Nevertheless, they reproduce to create one or multiple offspring.

Generation of chromosomes

The initial solutions are generated randomly and each solution is termed as the gene. The individual genes are incorporated as chromosomes and it is called the solution set. The numbers of genes are included with the chromosomes and the solution set for the population is created. The population of the genetic algorithm encompasses the chromosomes and the population size is activated as permanent. The numbers of solutions are activated in accordance with the typical genetic technique. In this case, the initial solutions are called the weight.

Cross over

In the cross over, the two parent chromosomes are chosen with the intention of exchanging their genes between them. The following example illustrates the parent chromosomes parent 1 and parent 2.

Shape of Parent 1 & 2

1	**2**	**3**	**1**	2	3	1	2

Parent 1

3	2	1	3	**2**	**1**	**3**	1

Parent 2

In parent 1 & 2 chromosomes, the bold lettered remain without any modification in their locations and the remaining gene of the chromosomes is exchanged between the parent chromosomes. Subsequent to the crossover, the chromosome takes the following shape.

Shape of new chromosome 1 & 2

1	2	3	1	**2**	**1**	**3**	2

New chromosome 1

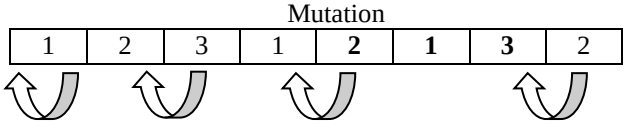
3	**2**	**3**	**1**	2	1	3	1

New chromosome 2

Mutation

Subsequent to the crossover, the new chromosome is transformed for augmenting the effectiveness of the solution and the bold depicts the transformed gene of the chromosome. In the novel mutation process, the matching order is selected within the offspring and it is exchanged from its position to other place for achieving the most brilliant optimal solution. The shift varying mutation approach is used in the mutation function and the orders of each chromosome are moved to leave one step and replaced by the new order. After the shift

the modifications within the off-spring are exhibited below.



Mutation process

From the above the gene of the off spring is moved one step left and the adapted new solution is achieved by the mutation procedure. The optimal solution is attained after completion of the mutation function and it illustrates the ultimate output of the outcome with their minimal optimized time, yielding the least make span duration.

Optimal solution

When the mutation function is completed, the new chromosomes are generated for the new solution sets. Later on, the fitness value is evaluated for the new solutions. The solution, which offers the best value, is shortlisted and is considered as the optimal solution. Otherwise, the processes mentioned above are repeated for the new solution sets.

In long run, we arrive at the optimal path from the hybrid GSO with AGA algorithm. Now each optimal path is home to a number of optimal nodes. If a new task emerges for the allocation of the resource, the innovative technique employs the optimal path in which the best nodes are shortlisted for the new task. With an eye on selecting the best nodes, the performed techniques deploy the resource cost, time and memory capacity.

3.3 Find the best test cases

These optimized features are used to find the best test cases these test cases are classified with the aid of kernel based SVM to predict the defects

3.4 Classification using hybrid kernel based support vector machine

Once the feature reduction is formed, the classification will be done based on the Hybrid kernel based Improved SVM (ISVM) classifier. In this classification, the optimal kernel is identified using Grey Wolf Optimization (GWO).

3.4.1. Kernel based support vector machine

In software defect prediction research optimized test cases are classified with the *aid* of kernel SVM. There are two very important phases in the SVM procedure such as the preparation phase and the effortless stage.

Training phase: Currently, the output of attribute choice is provided as the input of the preparation stage. The input utility supplies the group of values which cannot be alienated. Approximately each one of the probable isolation of the position places are comprehend by a hectic plane. In the Lagrange pattern, it is probable to put the partition of the hectic plane standard vector during the divergent kernel task. In this association, a kernel symbolizes a few tasks, which communicate to a dot product for definite kind of attribute recording. Yet, recording a position into a better quality dimensional gap is probable to direct to unnecessary assessment period and enormous storage requirements. By the outcome, in concrete perform, an original kernel task is initiated which is competent of openly estimating the dot product in the better-quality dimensional gap. The frequent edition of the kernel task is provided as follows.

$$K(U,V) = \varphi(U)^T \varphi(V) \quad (23)$$

In this view, the majority broadly engaged kernel tasks contain the linear kernel, Polynomial kernel, Quadratic kernel, Sigmoid and the Radial Basis task. Specified beneath are the terms for the different kernel task. For Linear Kernel:

$$linear_k(U,V) = u^T v + c \quad (24)$$

Where, u,v represents the inner products in linear kernel and c is a constant. For Quadratic Kernel:

$$quad_k(U,V) = 1 - \frac{\|u-v\|^2}{\|u-v\|^2+c} \quad (25)$$

Where, u,v - are the vectors of the polynomial kernel function in the input space. For Polynomial Kernel:

$$poly_k(U,V) = (\lambda u^T v + c)^e, \lambda > 0 \quad (26)$$

For Sigmoid Kernel:

$$sig_k(U,V) = \tanh(\lambda u^T v + c), \lambda > 0 \quad (27)$$

The effectiveness of the SVM consistently oriented on the variety of the kernel. In the original procedure, an original KSVM is predicted, dedicated for the noteworthy development in the categorization system. The mutual kernel task is

successfully engaged in the KSVM and the standard of the kernel task, $avg_k(U,V)$ is delivered beneath.

$$avg_k(U,V) = \frac{1}{2}\left(lin_k(U,V) + quad_k(U,V)\right) \quad (28)$$

$$avg_k(U,V) = \frac{1}{2}\left((u^T v + c) + \left(1 - \frac{\|u-v\|^2}{\|u-v\|^2+c}\right)\right) \quad (29)$$

By merging of two outcomes, the standard of the outcome is accomplished and developed to classification.

Testing phase: In the testing phases productivity from the classification choice is provided as to the experiment stage and the productivity specifies the subsistence or else the absence. In our proposed research we introduce an optimal GSO-GA based fault prediction in software testing to assess the cost effectiveness of test effort. Software testing plays a vital role in software development.

4. Experimental results with comparative analysis

The experimental result of GSO-GA algorithm is discussed below. The proposed system is implemented in Java (jdk 1.7) that has system configuration as core2duo processor with clock speed of 2.3GHZ and RAM as 2GB and that runs Windows 7 OS.

4.1 Performance evaluation

Performance evaluation is an adaptable tool used to: Measure actual performance against expected performance.

From table 1, the results of the fitness value using GSO-GA search measures are graphically represented in Fig. 2. In iteration 5, the fitness value is 40.46, in iteration 10 the fitness value is 17.21, in iteration 15 the fitness value is 6.96, in iteration 20 the fitness value is 3.337then in 25 iteration the fitness values is 0.0061.

Table 1. Fitness value depending on iteration

No of Iterations	Fitness value using GSO-GA
5	40.46
10	17.21
15	6.96
20	3.337
25	0.0061

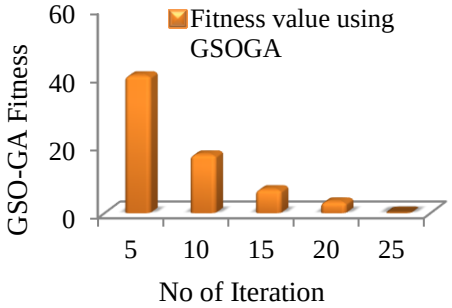


Figure.2 Graphical representation for fitness value using GSO-GA

Table 2. Time measures taken based on iteration

No of Iterations	Time (in ms)
5	557
10	592
15	620
20	629
25	667

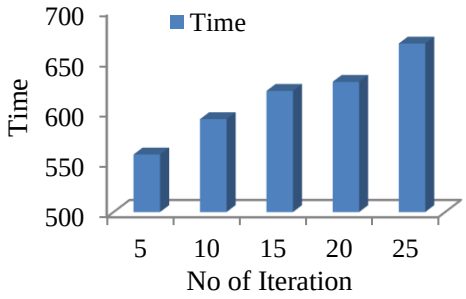


Figure.3 Graphical representation for Time

Table 3. Memory measures taken based on iteration

No of Iterations	Memory (in byte)
5	3540118
10	3516608
15	3583469
20	3595145
25	3563611

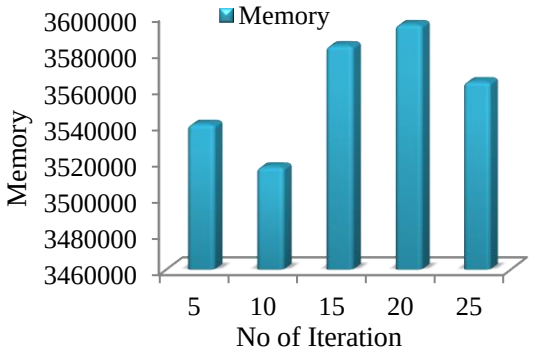


Figure.4 Graphical representation for memory

From Table 2, the results of the time are graphically represented in Fig. 3. The time taken in iteration 5 is 557, then in iteration 10 the time taken for that is 592 then in iteration 15 the time taken for that is 620, then in iteration 20 the time taken for that is 629, then in iteration 25 the time taken for that is 667.

From Table 3, the results of the memory are graphically represented in Fig. 4. The memory taken in iteration 5 is 3540118, then in iteration 10 the memory taken for that is 3516608, then in iteration 15 the memory taken for that is 3583469, then in iteration 20 the memory taken for that is 3595145 and in iteration 25 the memory taken for that is 3563611.

4.1.1. Average percentage of faults detected (APFD) metric

The APFD is computed by taking the weighted average of the percentage of faults detected during the implementation of the test suite. APFD values range from 0 to 100; higher values imply faster (better) fault detection rates. APFD can be computed as follows:

$$APFD = 1 - \left\{\frac{(Tf_1 + Tf_2 + \cdots Tf_m)}{mn}\right\} + \left({}^{1}/_{2n}\right) \quad (30)$$

Where n be the no of test cases and *m* be the no of faults. (*Tf1,….,Tfm*) are the position of first test *T* that exposes the fault.

Table 4. APFD results to be taken based on iteration

No of Iterations	APFD
5	65%
10	70%
15	67%
20	74%
25	83%

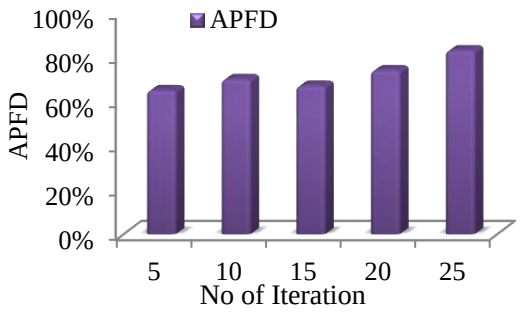


Figure.5 Graphical representation of APFD Evaluation Measures

Table 5. PTR results to be taken based on Iteration

No of Iterations	PTR
5	75%
10	78%
15	69%
20	76%
25	87%

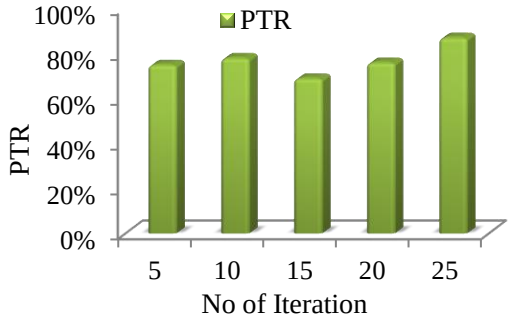


Figure.6 Graphical representation of PTR Evaluation Measures

From Table 4, the results of APFD are graphically represented in Fig. 5. In first iteration 5 the percentage of APFD is 65%, in second iteration 10 the percentage of APFD is 70%, in the third iteration 15 the percentage of APFD is 67%, then in fourth iteration 20 the percentage of APFD is 74% then the fifth iteration is 25 the percentage of APFD is 83%.

4.1.2. Problem tracking reports (PTR) metric

$$PTR(t,p) = {}^{nd}/_{n} \quad (31)$$

Let *t* - be the test suite under assessment,

n - The total number of test cases in the total number of test cases required to identify all faults in the program under test *p*.

4.2 Comparative analysis

The existing review works are compared in this section with the proposed work to show that our proposed work is better than the state-of-art works. We can establish that our proposed work helps to attain very good accuracy for the estimation of database using Improved Group search optimization-Genetic algorithm (GSO-GA). The Comparison outcomes are presented in the following Table 6.

The improved good outcomes of our proposed GSO-GA work are compared to existing GSO. The time measures of the existing method and proposed method to be measured is based on iteration. When we compare our result to the existing results our

proposed GSO-GA has given better result by taking minimum time to perform a process.

Table 6. Comparison for proposed and existing time measures

No of Iteration	Proposed GSO-GA (in ms)	Existing GSO (in ms)
5	557	628
10	592	658
15	620	698
20	629	765
25	667	678

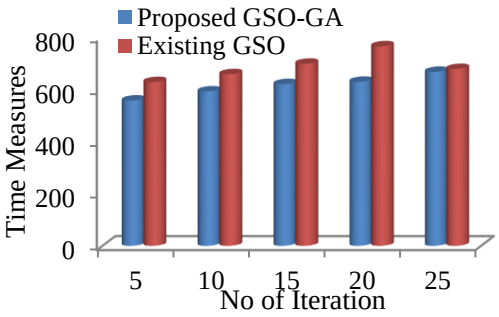


Figure.7 Graphical representation of proposed and existing time comparison measures

Table 7. Comparison for proposed and existing memory measures

No of Iteration	Proposed GSO-GA (in Byte)	Existing GSO (in Byte)	Existing GA (in byte)
5	3540118	3679564	3789567
10	3516608	3656585	3689548
15	3583469	3658952	3789562
20	3595145	3629574	3657894
25	3563611	3589745	3689524

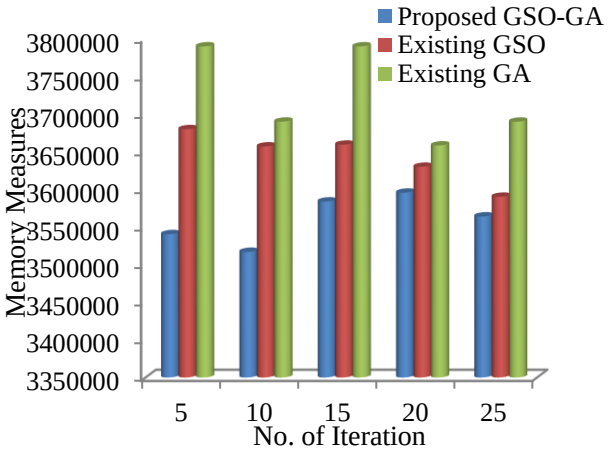


Figure.8 Graphical representation for proposed and existing memory comparison measures

Table 8. Comparison for proposed and existing accuracy measures

No of Iteration	Proposed KSVM	Existing SVM	Existing GA-NN
5	95	71	92
10	96	72	93
15	92	63	97
20	93	74	82
25	98	63	80

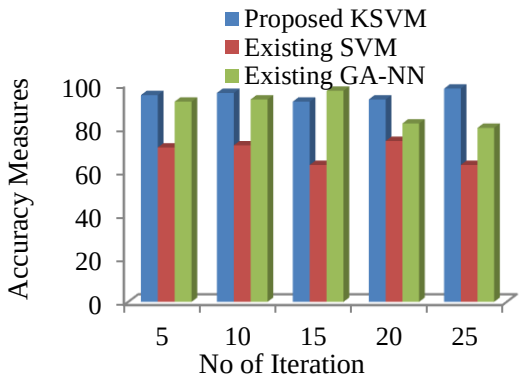


Figure.9 Graphical representation for proposed and existing accuracy measures

The memory measures of the existing method and proposed method is based on iteration. When we compare our result to the existing results our proposed GSO-GA has given a better result. It has taken a minimum byte of memory to perform an operation.

4.2.1. Classification accuracy comparison

The classification accuracy measures of the existing method and proposed method is based on iteration.

When we compare our result to the existing results, our proposed GSO-GA has given better classification accuracy. The existing work used the algorithm GA-NN to get 88% accuracy and SVM algorithm to get 68% accuracy. When we compare these two results to our proposed Kernel SVM we get 94% accuracy.

5. Conclusion

An Improved hybrid classification technique based fault prediction and classification with four phases, like feature extraction, feature optimization by using GSO-GA, find the best test cases and classification by using kernel based SVM, was proposed in this paper. The test cases are produced from the application program. The features are extracted from the test cases and then by utilizing the GSO-GA algorithm the features are optimized. These optimized features are used to find the best

test cases. These best test cases are classified with the aid of kernel based SVM to predict the defects. From the outcomes, we have showed that the kernel SVM classification algorithm utilized in our proposed work outperforms the classification with very good accuracy. Thus, we can observe that our proposed work is better than other existing works for the software fault prediction. In this paper, we have proposed kernel based SVM for classification GSO-GA for feature extraction as a proposed technique is compared against GSO. Results of proposed technique are in terms of APFD, PTR, Time, and Memory. We cannot promptly apply these results to other software systems different from the ones we employed. To address this issue, in the future we intend to replicate the performed analysis using other datasets. This is the only way to get better confidence on the generalizability of the results.

References

[1] V. Jayaraj and N. S. Raman, "An Hybrid Multilayer Perception Using GSO-GA For Software Defect Prediction", *International Journal of Recent Trends in Science and Management*, pp. 119-132, 2016.

[2] S. Lessmann, R. Stahlbock, and S. F. Crone, "Genetic algorithms for support vector machine model selection", In: *Proc. of International Joint Conf. On Neural Networks*, pp. 3063-3069, 2006.

[3] J. Anderson, H. Do, and S. Salem, "Experience Report: Mining Test Results for Reasons Other Than Functional Correctness", In: *Proc. of International Conf. on Software Reliability Engineering*, pp. 405-415, 2015.

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

[5] C. Catal, "Software fault prediction: A literature review and current trends", *Expert Systems with Applications*, Vol. 38, No. 4, pp. 4626-4636, 2011.

[6] I. Arora, V. Tetarwal, and A. Saha, "Open Issues in Software Defect Prediction", *Procedia Computer Science*, pp. 906-912, Vol. 46, 2015.

[7] A. Monden, T. Hayashi, and S. Shinoda, "Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing", *IEEE Transactions on Software Engineering*, Vol. 39, No. 10, pp. 1345-1357, 2013.

[8] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary Optimization of Software Quality Modeling with Multiple Repositories", *IEEE Transaction on Software Engineering*, Vol. 36, No. 6, pp. 852-864, 2010.

[9] A. Joshua, "Improving Software Quality Using Two Stage Cost Sensitive Learning", In: *Proc. of International Conf. on Science and Research*, pp. 1726-1728, 2013.

[10] C. Jin and S. W. Jin, "Software Reliability Prediction model based on support vector regression with improved estimation of distribution algorithms", *Applied Soft Computing*, Vol. 15, pp. 113-120, 2014.

[11] W. Li, Z. Huang, and Q. Li, "Three-way decisions based software defect prediction", *Knowledge Based Systems*, Vol. 91, pp. 263-274, 2016.

[12] X. Yang, K. Tang, and X. Yao, "A Learning-to-Rank Approach to Software Defect Prediction", *IEEE Transaction on Reliability*, Vol. 64, No. 1, pp. 234-246, 2015.

[13] L. Yu, "An evolutionary programming based asymmetric weighted least squares support vector machine ensemble learning methodology for software repository mining", *Information Sciences*, Vol. 19, pp. 31-46, 2015.

[14] H. M. Premalatha and C. V. Srikrishna "Software Fault Prediction and Classification using Cost based Random Forest in Spiral Life Cycle Model", *International Journal of Intelligent Engineering and Systems*, Vol. 11, No. 2, pp. 10-17, 2018.

[15] N. Singh, A. Kalpesh, and S. J. Narayanan, "Fault Prone Analysis of Software Systems Using Rough Fuzzy C-means Clustering", *International Journal of Intelligent Engineering and Systems*, Vol. 10, No. 6, pp. 1-8, 2017.

[16] V. Jayaraj, and N. S. Raman, "An Hybrid Multilayer Perceptron Using GSO-GA for Software Defect Prediction", In: *Proc. of 2nd International Conf. on Recent Trends in Engineering Science and Management*, pp. 119-132, 2016.

[17] S. D. Martino, F. Ferrucci, C. Gravino, and F. Sarro, "A Genetic Algorithm to Configure Support Vector Machines for Predicting Fault-Prone Components", In: *Proc. of International Conf. on Product Focused Software Process Improvement*, pp. 247-261, 2011.