Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 8, August 2019, pp. 2049-2061
DOI: 10.23940/ijpe.19.08.p4.20492061

Proposed Hybrid Approach to Predict Software Fault Detection

Manu Banga, Abhay Bansal, and Archana Singh*

ASET, Amity University, Noida, 201313, India

Abstract

The major challenge is to validate software failure dataset by finding unknown model parameters used. Previously, many attempts for software assurance were made using classical classifiers as Decision Tree, Naïve Bayes, and k-NN for software fault prediction. But the accuracy of fault prediction is very low as defect prone modules are very small as compared to defect-free modules. So, for solving modules fault classification problems and enhancing reliability accuracy, a hybrid algorithm proposed on Particle Swarm Optimization (PSO) & Modified Genetic Algorithm (MGA) for feature selection and Bagging for effective classification of defective or non-defective modules in a dataset. This paper presents an empirical study on NASA Metric Data Program (MDP) datasets, using the proposed hybrid algorithm. Results showed that our proposed hybrid approach enhances the classification accuracy compared with existing methods.

Keywords: software reliability; software fault classification; model parameter estimation; modified genetic algorithm; support vector machines; particle swarm optimization

(Submitted on June 22, 2019; Revised on August 3, 2019; Accepted on August 28, 2019)

1. Introduction

With the increasing demands of software, ranging from integral part of all advancements, protecting software from failure is critical to effective utilization of staff, cost and time as software defect can cause failure thereby preventing for achieving required software quality. Software has been playing a crucial role in manufacturing, security, finance, government, business and a lot more but still fails at being trustworthy at all times. So, software defect prediction system should predict defects possibility automatically. Among protective measures, the first concern is fault diagnosis in the software datasets, an important part of overall software reliability and maintenance system. When a fault occurs in software, the development team amounts of software metrics used to the dispatching software for analysis and designing. Software Trustworthiness is a critical part of each and every software modules since some software crashes due to defects at variable, methods, class initialization, However, many of these faults have intricately related connections as some faults remained unidentified or some faults added during error removal, making it difficult to detect the type of fault generated due to imperfect debugging or error generation. Therefore, it is necessary to use the most suitable fault classification algorithm to classify the collected data accurately for the best fault diagnosis. Software metrics are used for software defect prediction such as McCabe's and Halstead's metrics [1]. To compare results we are using NASA Metrics Data Programs datasets comprising 12 project data with an average number of the defective module is 20 percent but the PC2 dataset has the lowest percentage of 2.15 percent defected modules. This circumstance is referred to as "imbalanced data set", which is known to tremendously drop classification performance. The researchers [2-3] proposed several defect prediction model using classical available techniques in machine learning using the MDP datasets but ignored class imbalance issue, making their results biased and erroneous. Adopting a systematic approach, we propose a novel defect prediction system using feature selection for removing noisy and irrelevant attributes and bagging for class imbalance problem. After an empirical study, we compared our approach with various classifiers in terms of the measures Precision, Recall, F-Score, and Accuracy.

In this paper, we propose algorithms to improve the accuracy of software reliability estimation using feature selection by PSO-MGA (Particle Swarm Optimization-Modified Genetic algorithm) for reducing noise and selecting relevant

* Corresponding author.
E-mail address: archana.elina@gmail.com

attributes and thereby applying Bagging algorithm on relevant attributes and compared our results with various existing classification approaches. Our approach enables a classifier to classify defective modules with higher accuracy.

Following this, Section II presents a literature review of related work of defect prediction. Section III presents the Research Methodology used. Section IV presents Proposed Approach of hybrid algorithm aand in Section V, Experiment and Results on NASA Dataset are shown. It discusses results and analysis and Section VI discusses the conclusion of our research with its outcome.

Predictive Modelling for Software Defect Prediction

In Predictive Modelling of software fault data, learning is a critical step from the previous faulty dataset and estimating the optimal parameter values so that prediction could be accurate. Figure 1 shows software fault prediction using predictive modelling. [4] applied software maintenance techniques for defect prediction dataset thereby improving software quality and assurance for reducing maintenance cost incurred due to failures at a later stage.

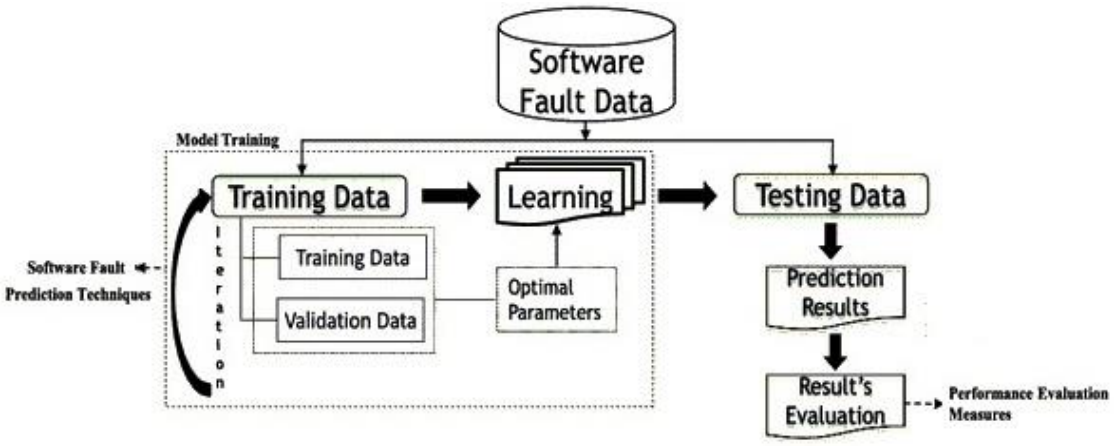

Figure 1. Software fault prediction using predictive modelling

2. Related Work Done

We surveyed the literature on software modules defects prediction using classification techniques available for improving software reliability. [5] worked on Software Metrics as measuring defects is a key for defect removal and defect prediction. He made a list of metrics and then used it for software defect prediction. [6] used Naïve Bayes classifier for predicting software defects achieved 71percent accuracy on the fault detection. [7] used Naïve Bayes classifier for defect detecting from static code. It compares its result with the entropy approach; he concluded Naive Bayes is effective classifier for static codes. [8] used Principal Component Analysis (PCA) to reduce multi-collinear complexity and achieved 10 percent higher accuracy in misclassification rate. [9] used classifiers for software assurance on digital communication system designing. [10] devised an Artificial Immune System on the Human immune system for defect prediction [11]. It extended Bayesian approach with Poisson distribution and proposed a novel software reliability model. [12] used Support Vector Machines for detecting errors in the NASA MDP data set obtained and achieved that modules having highest SVM scores in testing phase are more defect-prone. [13] used association rule mining for the learning phase and used the correlation coefficient for comparing results achieving the accuracy higher by 23 percent in the prediction rate. In the year, 2011 surveyed the performance of classification algorithm. He used NASA MDP datasets on twenty-two classifiers with accuracy outcome nearly same. [10] proposed a comprehensive approach in choosing classifiers for faulty datasets. [14] surveyed on the resampling techniques, threshold, and ensemble algorithms on Naïve Bayes, Random Forest and AdaBoost and he concluded that AdaBoost is effective classifier in achieving highest classification accuracy. Similarly, [15] used software metrics for predicting software module reuse and concluded by classifying reuse modules into four classes for fault prediction. [5] attempted to classify data using Naïve Bayes classifier for predicting defect at an early stage using Bayes Theorem and achieved significant results in fault detection at inductive learning. In the extension of the work in 2014, he proposed using evolutionary algorithms like Genetic Algorithms and modelled faults control rate using Mutation detection rate. [16] used design metrics for object-oriented software and used Naïve Bayes classifier for detecting faults at starting design phase as software development advances detecting faults is of high maintenance cost and time. (Nassif et al., 2019) used software metrics for predicting software module reuse and concluded by classifying reuse modules into four classes for fault prediction.

In existing research of software defect prediction, researchers focused on the classification of defective or non-defective modules but not on the selection of relevant attributes. After selecting relevant attributes classifier will focus only on useful attribute as required for research, thereby increasing classification accuracy. So we use feature selection based on PSO-MGA and then bagging for enhancing the accuracy of classifiers.

After conducting intensive research for defect classification and fault prediction, it has been observed as proposed methods ignored imbalance problem of classification. Our research focused on resolving imbalance problem by using bagging algorithm with feature selection. By feature selection, we get relevant attributes, so classification of defective or non-defective modules could be effectively and accurately done by a classifier.

3. Research Methodology

As dataset consists of noisy, irrelevant attributes so if using classification without feature selection may result in unnecessary waste of resources as faults if ignored could lead to failures which are very dangerous. Thereby, we perform Feature selection on NASA MDP datasets, which searches local solution to global solution by choosing only a relevant feature that are necessary to describe the software modules. This decreases the dimensionality and learning time of a classifier, and improves the accuracy of the classifiers in finding an optimal feature subset giving optimal performance in fault prediction [17]. It mainly has three types: Embedded Feature selection, Wrapper Feature Selection and Filter based Feature Selection. In Embedded Feature Selection, regularization of the dataset is done to avoid over fitting of variance. In filter-based feature, selection uses known characteristics of the data based on a metrics. In Wrapper based feature selection learning from the training, dataset is done and learning is used on test data for getting optimal parameters. So, we choose the wrapper based selection approach as it has the best accuracy in feature selection.

3.1. Modified Genetic Algorithm (*MGA*)

Modified Genetic Algorithm [18] is a search and optimization method that detects an approximate solution to process. It imitates the evolution in biology as a strategy for providing a solution to the problem of optimizing a population of a candidate with accuracy A , the eight of accuracy W_A is classifying defective or non-defective module of feature value F_i, C_i is a feature cost, and P is a constant value with towards a predefined fitness. The imitation process commences by processing the search operation with the help of set of solutions in a random manner, and then solution sets are coded in binary strings using fitness function, as shown in Equation (1).

$$fitness = W_A \times A + W_F \times \left(P + \left(\sum_{i=1}^{n_f} C_i \times F_i \right) \right)^{-1} \tag{1}$$

MGA includes four operators' viz., initialization, selection, crossover and mutation. The solution is obtained for each chromosome in the search space by actively operating MGA with the randomly-initialized population. It has the number of genes and the quality is measured depends on its fitness function and the representation of each rules adaptation. GA needs a conversion of design space into the genetic space. It uses a population of points single at a time, whereas the other methods use the single point approach. Thus, the GA is able to provide a number of solutions simultaneously during iteration steps. The MGA process starts with the random generation or selection of a set of solutions or chromosomes. The crossover and mutation techniques are applied for generating a new offspring or chromosome. The crossover technique splits two chromosomes and then combines the first portion of one chromosome with the second portion of the other chromosome. This technique performs flipping of one or more bits of a chromosome. The chromosomes are evaluated using fitness criteria. The chromosome having the highest fitness value is considered as the best solution. This is until iteration continues known as the generation of crossover and mutation occurs and the objective function is checked for a satisfactory solution. If stooping criterion defined achieved then best chromosome is selected otherwise iteration continues as shown in Figure 2.

3.2. Particle Swarm Optimization (*PSO*)

The Particle Swarm Optimization is used for achieving an objective function in $N-$dimensional space. It searches individual swarm in loop that is updated in iterations. p_{size} is size of population and for an optimal solution; it considers two values own previously best value p_{best}, and other members best value g_{best} so p_{best} is cognition part, and g_{best} is social part.[19] as shown in Figure 3.

The N-dimensional position for the particle is called candidate solution having a position $x^t{}_i$and $v^t{}_i$ at iteration t can be represented as

$$x_i^t = \{x_{i1}^t, x_{i2}^t, \cdots, x_{iN}^t\} \tag{2}$$

Where x_i^t represents each particle position

$$v_i^t = \{v_{i1}^t, v_{i2}^t, \cdots, v_{iN}^t\} \tag{3}$$

Where $v_i^t =$ represents each particle velocity

For obtaining the solution p_i^t that particle i obtained until iteration t.

$$p_i^t = \{\, p_{i1}^t,\ p_{i2}^t, \cdots,\ p_{iN}^t\} \tag{4}$$

The solution obtained from p_i^t in the population at iteration t is p_g^t.

$$p_g^t = \{p_{g1}^t, p_{g2}^t, \cdots, p_{gN}^t\} \tag{5}$$

According to g_{best} and p_{best} is V_{in}^t, c_1 value of is p_{best} and c_2 is value of g_{best}, w is adjusted weight value for tracking swarm values in (0,1)

$$V_{in}^t = w \times V_{in}^{t-1} + c_1 r_1 (p_{in}^t - x_{in}^t) + c_2 r_2 \left(p_{gn}^t - x_{in}^t\right) \tag{6}$$

Potential solution X_{in}^{t+1} obtained after each particle movement

$$X_{in}^{t+1} = X_{in}^t + V_{in}^t \tag{7}$$

Step 1: New population is initialized.
Step 2: Chromosomes are randomly produced from this new population.
Step 3: Evaluation of each chromosome is done with the help of fitness function defined for their probability in early phase.
Step 4: With the help of selection operator, the algorithm selects chromosomes in a random way.
Step 5: Offspring chromosomes are generated by applying genetic operators such as crossover and mutation.
Step 6: If the end condition such as iteration is maximum is achieved then MGA process stops. Otherwise step 4 is repeated for further solutions.

Figure 2. MGA Algorithm for feature selection [18]

Step 1 Initialization of candidate solution with its position value and velocity value.
Step 2 Evaluate fitness value of each particle.
Step 3 Update p_{best} personal fitness value.
Step 4 Update g_{best} global fitness value.
Step 5 If end condition such as iteration is maximum is achieved then PSO process stops with Global best solutions else update position value and update velocity value. Otherwise step 3 and step 4 are repeated for further solutions.

Figure 3. Algorithm particle swarm optimization [19]

3.3. Bagging

The application of bagging is used to improve the results of classification techniques used. The technique Bootstrap results in z fitted models. The models are brought together to apply regression or generate random selection of models. The bagging classifier segregates z datasets into z' training sets. The model is built with the help of z' dataset and voting for each model results in the model, reducing variance and evading over fitting of the model. For improving classification, Bagging [18] technique is used.

Input: Training Original Dataset M of size z

Apply Bootstrap Sampling on S generating modified dataset as M' of size z'.

Case 1: If $(z' < z)$, by uniform sampling of S with replacement, probability of repeating some samples in M'.

Case 2: If $(z' = z)$ then for M' is expected to have $\left(1 - \frac{1}{e}\right)$ unique samples, rest being repeating samples.

4. Proposed Approach-Software Defect Prediction Framework

In the proposed approach, the hybrid of algorithms, shown in Figure 4, is used to improve the accuracy of software reliability estimation using feature selection by PSO-MGA (Particle Swarm Optimization-Modified Genetic algorithm) to reduce noise and select relevant attributes. The integration of both techniques, PSO-MGA is used for feature selection.

1. Initialize the maximum iteration M_{max} and P features parameter value randomly in PSO.
2. Calculate features value, saving the place of feature $g_{best,PSO}$ of least objective function.
3. Split features into two groups, with probability$H$of size as $P * E$. One group uses PSO for position updating and other group uses MGA.
4. Group using PSO update their position and velocity according to Equation (6) and (7) for obtaining potential solution.
5. Using the above equation, a new objective function is obtained, G_{PSO}. The group using MGA works on three operators: selection, crossover and mutation. Based on fitness function value, selection operator chooses features. Using crossover new generation of attributes. The crossover technique splits two attributes and then combines the first portion of one attributes with the second portion of the other attributes. This technique performs flipping of one or more bits of a attributes. The attributes are evaluated using fitness criteria. The attributes having highest the fitness value is considered as the best solution, $g_{best,MGA}$ till iteration continues known as the generation of crossover and mutation occurs and the new objective functionG_{GA} is obtained.
6. Compare $G_{PSO}$and G_{MGA}
   if$G_{PSO} < G_{GA}$
   global best solution is $g_{best,PSO}$
   $g_{best,MGA}$is replaced by $g_{best,PSO}$and $G_{MGA}$is replaced by G_{PSO}
   else
   global best solution is $g_{best,MGA}$
   $g_{best,PSO}$is replaced by $g_{best,MGA}$and $G_{PSO}$is replaced by G_{MGA}
7. Best Solution is obtained, feature selected from NASA MDP Dataset on the basis above steps.

Figure 4. Hybrid PSO-MGA algorithm

4.1. Hybrid PSO-MGA Algorithm

Figure 5 depicts the working of above algorithm.

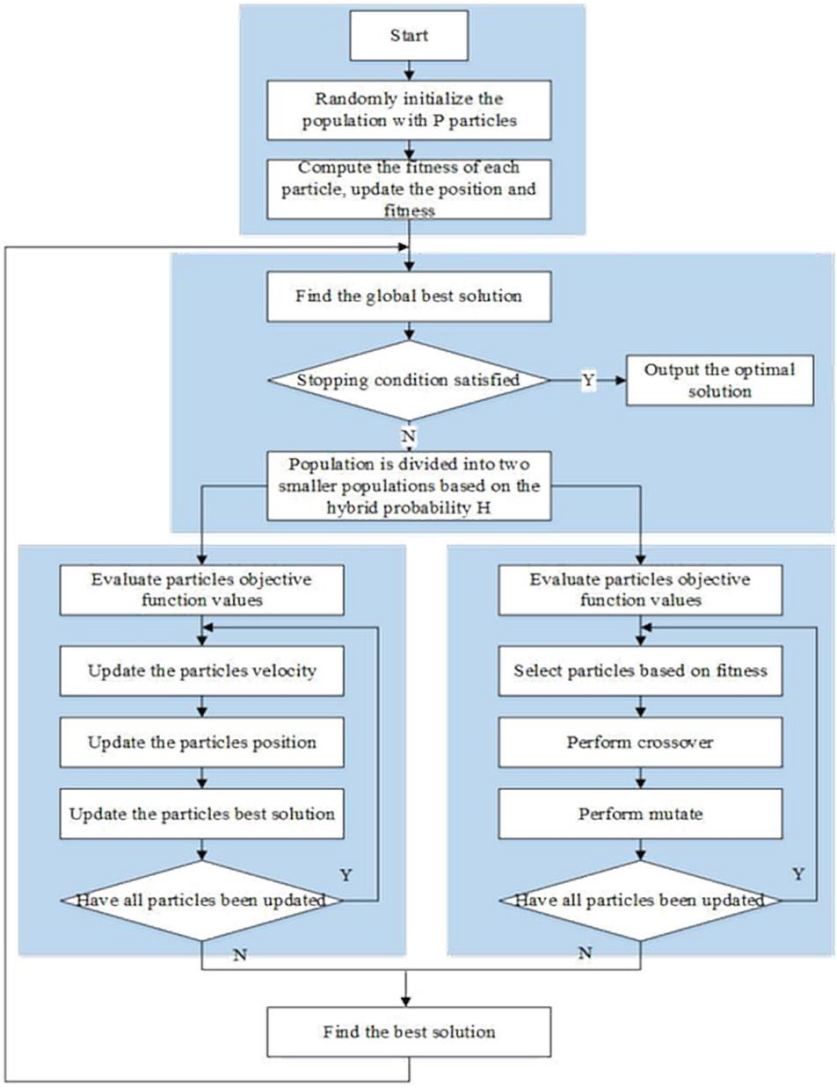

Figure 5. PSO-MGA Hybrid Algorithm for feature selection

4.2. Applying of Bagging with Feature Selection PSO-MGA Algorithm

For avoid overtraining in PSO-MGA hybrid algorithm, we are integrating bagging for resolving class imbalance problem of defective or non-defective modules. Using Bootstrap samples of NASA MDP datasets in m corresponding models that are refitted and combined by voting. Bagging random sampling is performed and the classifier splits a training dataset$m$ into m'new training sets. Thus, a new model on new training dataset is made, and voting results of m' models are combined to obtain final classification results for getting optimized feature sets. Figure 6 shows the integration of Bagging with hybrid algorithm of feature selection.

5. Experiment and Results

5.1. Data Sets Used

We use NASA Metric Data Program (MDP) [7] dataset from PROMISE SOFTWARE ENGINEERING REPOSITORY that has 12 projects data as CM1, KC1, KC2, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, of NASA spacecraft software. We study each project with their unique attribute. In Table I, 22 attribute information of KC1 project is given. It has 1183 samples with defected class as 314 and non-defected class 869, percentage of defected class is 26.54. In KC3 project has 194 samples, 40 attributes, with defected class as 314 and non-defected class 158, percentage of defected class is 66.53. In CM1 project has 327 samples, 38 attributes, with defected class as 42 and non-defected class 285, percentage of defected class is 12.84. In KC2 project has 7782 samples, 22 attributes, with defected class as 1672 and non-defected class 6110,

percentage of defected class is 21.49. In MC1 project has 1988 samples, 39 attributes, with defected class as 46 and non-defected class 1942, percentage of defected class is 2.31. In MC2 project has 125 samples, 40 attributes, with defected class as 44 and non-defected class 81, percentage of defected class is 35.20. In MW1 project has 253 samples, 38 attributes, with defected class as 27 and non-defected class 226, percentage of defected class is 10.67. In PC1 project has 705 samples, 38 attributes, with defected class as 61 and non-defected class 644, percentage of defected class is 8.65. In PC2 project has 745 samples, 37 attributes, with defected class as 16 and non-defected class 729, percentage of defected class is 2.15. In PC3 project has 1077 samples, 38 attributes, with defected class as 134 and non-defected class 943, percentage of defected class is 12.44. In MC2 project has 125 samples, 40 attributes, with defected class as 44 and non-defected class 81, percentage of defected class is 35.20. In PC4 project has 1287 samples, 38 attributes, with defected class as 177 and non-defected class 1110, percentage of defected class is 13.75. In PC5 project has 1711 samples, 39 attributes, with defected class as 471 and non-defected class 1240, percentage of defected class is 27.53. From 12 projects, we found defect percentage as 20% with maximum defect percentage in KC3 as 66.53% and minimum defect percentage as 2.15% in PC2.

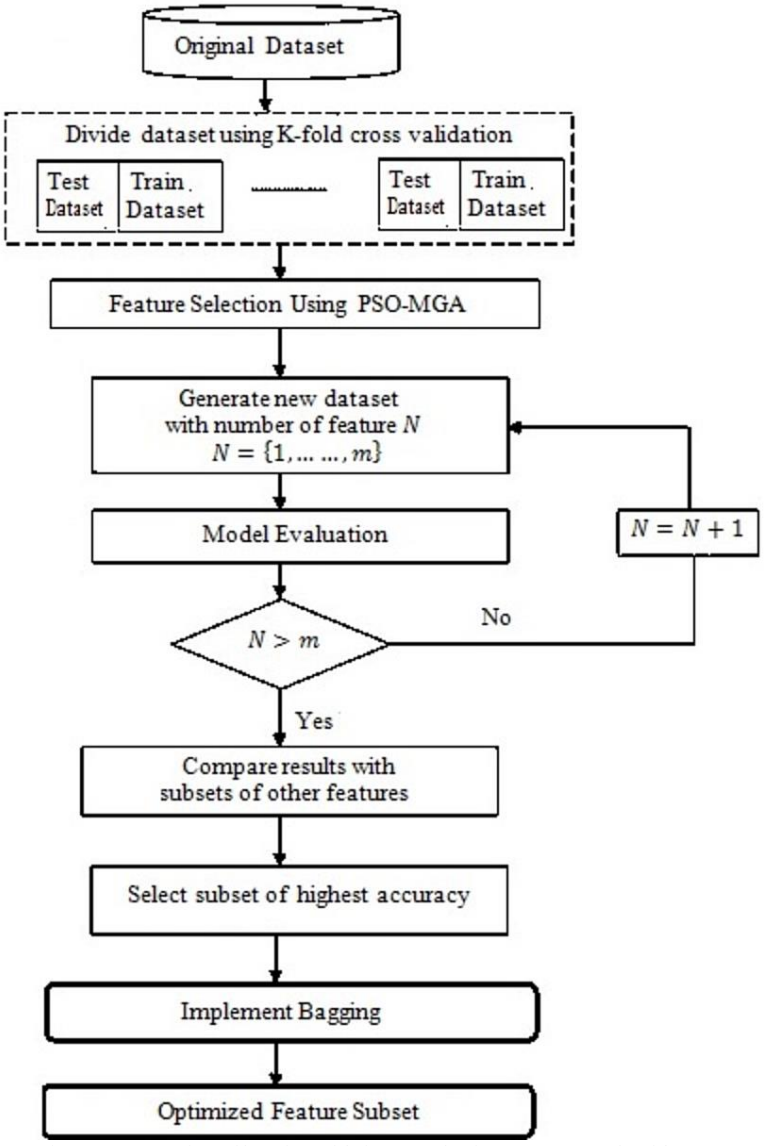

Figure 6. Integration of bagging with feature selection PSO-MGA algorithm

5.2. Model Validation

For validation of the approach, we divide dataset into *k*-fold cross-validation for learning and testing data. The value of k depends on the group in which dataset has to be split for checking accuracy of a model on unseen or test data. The first fold is treated as validation set and used to fit the remaining *k*-1 folds. We use 10-fold cross validation as it generally results in less biased or optimized estimate of the model [20].

5.2.1. Performance Measure

The classifier performance is depicted in two-dimension by the ROC curve. For comparison of classifiers, the ROC performance must be represented in a single scalar vector of expected performance. The commonly used technique is to calculate the area under the ROC curve. The area under the curve of a classifier is equivalent to the probability that the

- Instance is positive and classified as positive – True Positive (TP)
- Instance is positive and classified as negative – False Negative (FN)
- Instance is negative and classified as negative – True Negative (TN)
- Instance is negative and classified as positive – False Positive (FP)

The true positive rate of a classifier is defined as a percentage of examples correctly predicted as non- defective against all of the actually non-defective

$$True\ Positive\ Rate(t_p) = \frac{Positive\ correctly\ classified}{Total\ Negative} = \frac{TP}{TP + FP} \tag{8}$$

The false positive rate of a classifier is defined as a percentage of examples correctly predicted as non- defective against all of the actually non-defective

$$True\ Negative\ Rate(f_p) = \frac{Negative\ correctly\ classified}{Total\ Negative} = \frac{FN}{FN + TN} \tag{9}$$

The sensitivity of a classifier is a percentage of examples correctly predicted as defective against all of the actually defective

$$Sensitivity = recall = \frac{Total\ Positive\ classified}{Total\ samples} = \frac{TP}{TP + FN} \tag{10}$$

The specificity of classifiers is a percentage of examples negatively predicted as defective against all of the actually defective

$$Specificity = \frac{Total\ Negative\ classified}{Total\ samples} = \frac{TN}{TP + FN} \tag{11}$$

The Accuracy of a classifiers defined as percentage of positively predicted over all the predicted values.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

The Precision of a classifiers defined as percentage of examples predicted as defective against all of the predicted defective

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

$\mathrm{F} - \mathrm{measure}$ of a classifier defined as a weighted harmonic mean of precision and recall

$$F - measure = 2.\frac{Precision.Recall}{Precision + Recall} \tag{14}$$

5.2.2. Model Evaluation

For evaluating the performance of classifiers used on our model between interval $\{0.50, 1\}$ as:

The performance of the classifiers is categorized as follows:

- 0.90: excellent
- 0.80 - 0.90: good
- 0.70 - 0.60: fair
- 0.60 - 0.70: not acceptable
- 0.50 - 0.60: failure

5.2.3. Model Comparison using t −Test

We used t −Test for comparing two samples of statistical differences between paired values of two samples. We obtain a unique value after considering the variation of values within each sample. We used significance level denoted as α, which signifies the probability of rejecting the null hypothesis if it is true. We choose a significance level $\propto = 0.05$.

5.3. Results and Analysis

In the experiments, the classifiers are K-Nearest Neighbour (KNN), Random Forest (RF), Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM) and NASA metrics data program (MDP). Dataset and results are shown in Table 1.

Table 1. Accuracy of classifiers on NASA MDP datasets (without feature selection & bagging)

Classifiers	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
KNN	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
RF	0.573	0.485	0.477	0.525	0.74	0.618	0.649	0.678	0.2
SVM	0.753	0.752	0.642	0.761	0.714	0.79	0.534	0.75	0.79
LSTSVM	0.734	0.786	0.67	0.739	0.732	0.781	0.811	0.756	0.838
MS-LSTSVM	0.713	0.791	0.647	0.71	0.625	0.784	0.918	0.79	0.833

After feature selection (PSO-MDP) on NASA metrics data program (MDP) dataset, we conducted experiments on NASA MDP data sets using classifiers K-Nearest Neighbour (KNN), Random Forest (RF), Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), and Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM). Result are shown in Table 2.

Table 2. Accuracy of classifiers on NASA MDP datasets (without feature selection & bagging)

Classifiers	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
KNN	0.668	0.5	0.67	0.78	0.65	0.73	0.554	0.645	0.75
RF	0.573	0.485	0.477	0.525	0.741	0.696	0.904	0.738	0.601
SVM	0.753	0.752	0.642	0.761	0.659	0.774	0.139	0.47	0.89
LSTSVM	0.734	0.786	0.67	0.739	0.724	0.799	0.811	0.75	0.861
MS-LSTSVM	0.713	0.795	0.647	0.761	0.742	0.852	0.822	0.81	0.903

After feature selection (PSO-MDP) and Bagging implementation on NASA metrics data program (MDP) dataset, we conducted experiments on NASA MDP data sets using classifiers K-Nearest Neighbour (KNN), Random Forest (RF), Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), and Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM). Results are shown in Table 3.

Table 3. Accuracy of classifiers on NASA MDP datasets (with feature selection and bagging)

Classifiers	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
KNN	0.768	0.789	0.657	0.786	0.65	0.73	0.554	0.645	0.75
RF	0.463	0.697	0.477	0.525	0.741	0.696	0.904	0.738	0.601
SVM	0.721	0.725	0.642	0.761	0.659	0.774	0.139	0.47	0.895
LSTSVM	0.756	0.786	0.867	0.739	0.724	0.799	0.811	0.75	0.861
MS-LSTSVM	0.617	0.95	0.547	0.761	0.742	0.852	0.822	0.81	0.903

For observing statistical difference in performance of classifiers K-Nearest Neighbour (KNN), Random Forest (RF), Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), and Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM), we conduct t −test on each classifier without feature selection and bagging. Results are shown in Table 4.

For observing statistical difference in performance of classifiers K-Nearest Neighbour (KNN), Random Forest (RF), Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), and Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM), we conduct t −test on each classifier with feature selection. Results are shown in Table 5.

Table 4. T-TEST of Classifiers without feature selection and bagging

Classifiers	Value of *t*-test	Result
KNN	0.23	Not Significant
RF	0.3	Significant
SVM	0.005	Significant
LSTSVM	0.008	Significant
MS-LSTSVM	0.004	Significant

Table 5. T-TEST of with feature selection

Classifiers	Value of *t*-test	Result
KNN	0.423	Not Significant
RF	0.002	Significant
SVM	0.008	Significant
LSTSVM	0.005	Significant
MS-LSTSVM	0.005	Significant

For observing statistical difference in performance of classifiers K-Nearest Neighbour (KNN), Random Forest (RF), Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), and Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM), we conduct t −test on each classifier with feature selection and bagging. Results are shown in Table 6.

Table 6. T-TEST with feature selection & bagging

Classifiers	Value of *t*-test	Result
KNN	0.323	Not Significant
RF	0.003	Significant
SVM	0.007	Significant
LSTSVM	0.004	Significant
MS-LSTSVM	0.005	Significant

The various classifiers K-Nearest Neighbour (KNN), Random Forest (RF), Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), and Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM) with feature selection and bagging model are validated by comparing parameters value Precision, Recall, F-Measure, Accuracy. Results are shown in Table 7 and Figures 7-10.

Table 7. Performance measures of different classifiers

Parameters	KNN	RF	SVM	LSTSVM	MW-LSTSVM
Precision	74.54	78.42	81.56	82.84	86.61
Recall	71.7	75.38	81.46	82.46	88.64
F-Measure	73.98	77.74	81.89	82.93	87.83
Accuracy	75.83	81.96	84.56	82.46	89.85

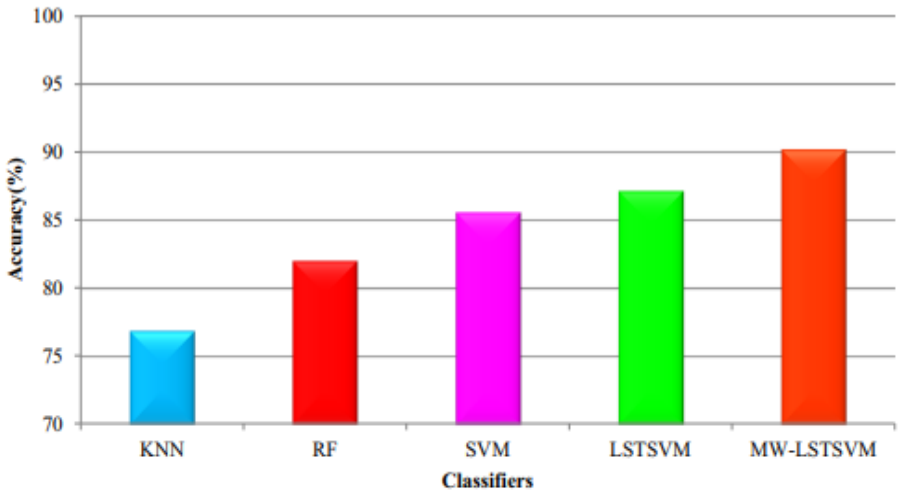

Figure 7. Accuracy of classifiers comparison

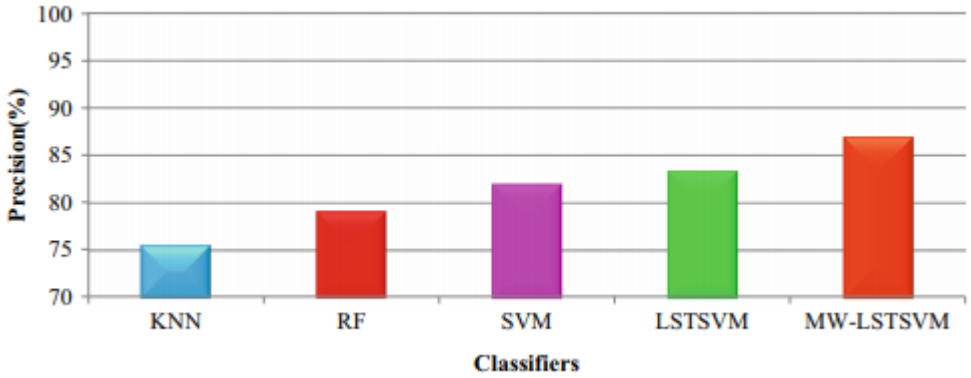

Figure 8. Precision of classifiers comparison

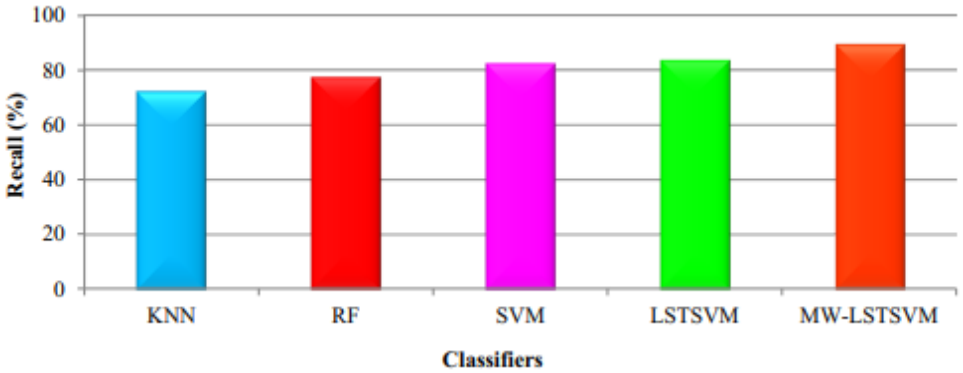

Figure 9. Recall of classifiers comparison

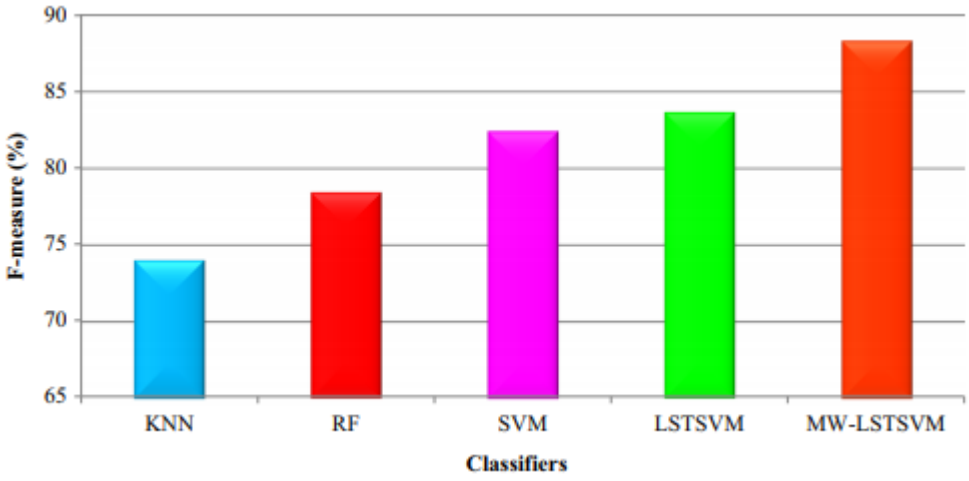

Figure 10. F-Measure of classifiers comparison

6. Conclusions and Future Scope

In this paper, the attempt was made to find software defects using hybrid approach. Identifying the defects is the most crucial step as defect-prone software could lead to hazard, which leads to loss of reliability of the software. So, the defects detected and classified better to the system but the available research work ignores the class imbalance issue. The avoidance of the class imbalance problem leads to decrease in the accuracy of the classifier. In this paper, we have used hybrid PSO-MGA for feature selection and Bagging Techniques for increasing classification accuracy of defective or non-defective modules on different classifiers K-NN, RF, SVM, LSTSVM, and MW-LSTSVM. The experimental results show that the MW-LSTSVM classification algorithm has achieved highest accuracy among all other classifier with 91.3% accuracy, thus increasing the efficiency by 9.8% than existing approaches in classifying defective or non-defective modules. In our future work, this accuracy can be extended by using deep learning techniques to better feature engineering out of the datasets, which leads to more accuracy of the classifying results. This will thereby improve software reliability.

Annexure

Details of software metrics of NASA Project KC1 Dataset Description.

S.No.	Attribute	Description
1	LOC	For Counting line of code in a software module
2	v(g)	For Measuring McCabe Cyclomatic Complexity
3	iv(g)	For Measuring McCabe Design Complexity
4	ev(g)	For Measuring McCabe Essential Complexity
5	N	Total Number of operators and operands
6	V	Volume
7	L	Program Length
8	D	Measure Difficulty
9	I	Measure Intelligence
10	E	Measure Effort
11	B	Effort Estimate
12	T	Time Estimate
13	Locoed	Number of lines in a software module
14	Locomments	Number of Comments
15	Loblank	Number of blank lines
16	Locode and comment	Number of code and comments
17	Uniq_op	Unique Operators
18	uniq_opnd	Unique Operands
19	total_op	Total Operators
20	total_opnd	Total Operands
21	Branch count	Number of branch counting
22	Defects	Classes defining software module having defect or not

References

1. T. M. Khoshgoftaar, N. Seliya, and Y. Liu, "Genetic Programming-based Decision Trees for Software Quality Classification," in *Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence*, pp. 374-383, IEEE, November 2003
2. T. Wang and W. H. Li, "Naive Bayes Software Defect Prediction Model," in *Proceedings of 2010 International Conference on Computational Intelligence and Software Engineering*, pp. 1-4, IEEE, December 2010
3. D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The Misuse of the NASA Metrics Data Program Data Sets for Automated Software Defect Prediction," in *Proceedings of 15th Annual Conference on Evaluation & Assessment in Software Engineering* (*EASE 2011*), pp. 96-103, IET., April 2011
4. V. U. B. Challagulla, F. B. Bastani, I. L. Yen, and R. A. Paul, "Empirical Assessment of Machine Learning based Software Defect Prediction Techniques," *International Journal on Artificial Intelligence Tools*, Vol. 17, No. 2, pp. 389-400, 2008
5. D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme, "Preliminary Comparison of Techniques for Dealing with Imbalance in Software Defect Prediction," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, pp. 43, ACM, May 2014
6. K. O. Elish and M. O. Elish, "Predicting Defect-Prone Software Modules using Support Vector Machines," *Journal of Systems and Software*, Vol. 81, No. 5, pp. 649-660, 2008
7. Y. Jiang, B. Cuki, T. Menzies, and N. Bartlow, "Comparing Design and Code Metrics for Software Quality Prediction," in *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, pp. 11-18, ACM, May 2008
8. D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme, "Preliminary Comparison of Techniques for Dealing with Imbalance in Software Defect Prediction," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, pp. 43, ACM, May 2014
9. C. Catal and B. Diri, "Investigating the Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction Problem," *Information Sciences*, Vol. 179, No. 8, pp. 1040-1058, 2009
10. M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the Nasa Software Defect Datasets," *IEEE Transactions on Software Engineering*, Vol. 39, No. 9, pp. 1208-1215, 2013
11. H. S. Shukla and D. K. Verma, "A Review on Software Defect Prediction," *International Journal of Advanced Research in Computer Engineering & Technology* (*IJARCET*), Vol. 4, No. 12, pp. 4387-4394, 2015
12. M. Hamill and K. Goseva-Popstojanova, "Exploring Fault Types, Detection Activities, and Failure Severity in an Evolving Safety-Critical Software System," *Software Quality Journal*, Vol. 23, No. 2, pp. 229-265, 2015
13. R. Malhotra, "A Systematic Review of Machine Learning Techniques for Software Fault Prediction," *Applied Soft Computing*, Vol. 27, pp. 504-518, 2015
14. W. Shuo and Y. Xin, "Using Class Imbalance Learning for Software Defect Prediction," *IEEE Transactions on Reliability*, Vol. 62, No. 2, pp. 434-443, 2013
15. A. B. Nassif, M. Azzeh, A. Idri, and A. Abran, "Software Development Effort Estimation using Regression Fuzzy Models,"

Computational Intelligence and Neuroscience, Vol. 5, 2019
16. G. Chandrashekar and F. Sahin, "A Survey on Feature Selection Methods," *Computers & Electrical Engineering*, Vol. 40, No. 1, pp. 16-28, 2014
17. T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Transactions on SE*, Vol. 33, No. 1, pp. 2-13, 2007
18. S. Jiang, K. S. Chin, L. Wang, G. Qu, and K. L. Tsui, "Modified Genetic Algorithm-based Feature Selection Combined with Pre-Trained Deep Neural Network for Demand Forecasting in Outpatient Department," *Expert Systems with Applications*, Vol. 82, pp. 216-230, 2017
19. D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Software Defect Prediction using Static Code Metrics Underestimates Defect-Proneness," in *Proceedings of the 2010 International Joint Conference on Neural Networks* (*IJCNN*), pp. 1-7, Barcelona, 2010
20. P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-Validation," Encyclopedia of Database Systems, Springer, Boston, MA, 2009