Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

Genetic algorithm-based multi-objective optimization model for software bugs prediction

Bakre Oluseye Musinat[1], *Femi* Temitope Johnson[2], *Olusegun* Folorunso[2], *Ihekwoaba* Ezinne[3]

1 – Department of Computer Science, Olabisi Onabanjo University, Ago-Iwoye, Ogun State, Nigeria
2 – Department of Computer Science, Federal University of Agriculture, Abeokuta, Ogun State, Nigeria.
3 –Computer Science Department, Federal College of Education (Technical), Akoka, Lagos State, Nigeria.

Corresponding author contact: femijohnson123@hotmail.com

Abstract.

The accuracy and reliability of softwares are critical factors for consideration in the operation of any electronic or computing device. Although, there exist several conventional methods of software bugs prediction which depend solely on static code metrics without syntactic structures or semantic information of programs which are more appropriate for developing accurate predictive models. In this paper, software bugs are predicted using a Genetic Algorithm (GA)-based multi-objective optimization model implemented in MATLAB on the National Aeronautics and Space Administration (NASA) dataset comprising thirty-eight distinct factors reduced to six (6) major factors via the use of the Principal Component Analysis (PCA) algorithm with SPSS, after which a linear regression equation was derived. The developed GA- based multi-objective optimization model was well-tried and tested. The accuracy and sensitivity level were also analyzed for successful bug detection. The results for optimal values ranging from 95% to 97% were recorded at an average accuracy of 96.4% derived through MATLAB-implemented measures of critical similarities. The research findings reveal that the model hereto proposed will provide an effective solution to the problem of predicting buggy software in general circulation.

Keywords: Software bugs, Genetic Algorithm, Optimization, Prediction, Machine learning

1 Introduction

Software bugs are usually known for the adverse effects they have on computer programs and devices, and for causing frequent issues of hardware malfunction. Predicting defective software modules has been a great challenge (Catal & Diri, 2009) although it helps detect defect-prone modules (Kim et al., 2011), which play critical roles in software quality assurance.

A software bug is simply an error or fault causing a program to generate wrong results or function abnormally. Several causes of bugs have been associated with errors in source program designs, compilation, and operating systems malfunctions with great glitches of related disasters (Gupta, Dharmendra, & Kavita, 2017). A high rate of death was recorded in 1980 due to code bugs in the Therac-25 radiation therapy system. Also, historical records of the events in 1994, detailing the reason for the Chinook Royal Air Force helicopter smash into the Kintyre Mall has been triggered by a software error in the engine control computer of the aircraft thereby killing twenty-nine (Simon, 2012). Similarly, The European Space Agency's $1 billion prototype Ariane 5 rocket after a minute launch was also destroyed due to the presence of bugs in its on-board guidance computer program (Okutan & Yıldız, 2014).

Nonetheless, present-day software defects prediction entails the use of machine learning (Bavisi, Mehta, & Lopes, 2014) with historical data in repositories (Catal et al., 2009). The outcome from this will help developers gain stability and reliability of apps to find and repair possible defects in softwares. Cross-project and within-project software defect prediction are common categories of software prediction processes (Rahman, Posnett & Devanbu, 2012), which could be adopted, either one or both, for software bug prediction depending on the homogeneity of the sourced data to the target data (Zhou et al., 2018).

The first step towards the process of debugging through machine learning is the collection of instances from software repositories such as problem tracking systems, version control systems, or e-mail

Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

repositories for the development of a prediction model. Each instance could be a source code file containing class, functions or methods, device or software package for classifying software as either defective or not non-defective (Naidu & Geethanjali, 2013). Software are considered defective when they contain several bugs interfering with their functionality (Shepperd, Bowes & Hall, 2014), such as causing a functional system to crash or stop responding to users' requests and commands. Security challenging bugs may provide malicious users access and unauthorized privileges to an organization's system (Akmel, Birihanu & Siraj, 2017). Pre-processing machine learning techniques (feature selection, data normalization, and noise reduction) could also be performed on the instances with labels and metrics (Shivaji et al., 2012; Kim et al., 2011). For accurate prediction, the model must be well-tried with lots of training instance samples and tested before being deployed for software bugs detection in real-life sceneries.

Adopted in the present paper, is an adaptive heuristic machine learning search (Genetic Algorithm), founded on the same principle as the natural fitness selection of genes, for a multi-variable input selection for a more optimized loss (parametric features) and cost (Line numbers) functions in bugs' prediction. The Genetic Algorithm (GA) model has been used extensively to solve software testing problems involving classification, optimization and regression. The remaining section of this paper is structured as follows: Section 2 commences with a brief introduction to the Genetic Algorithm, its operational stages towards software bugs prediction with optimization methods and Section 3 provides the methodology starting with data description and selection, preprocessing techniques and optimization methods. The implementation of the Genetic Algorithm-based multi-objective optimized model and the results are demonstrated in Section 4. Highlighted, lastly, in Section 5, are the conclusions reached and the future directions of research.

2 Genetic Algorithm for optimization in software bugs prediction.

The genetic algorithm (GA) developed by John Holland in 1960 (Bies, et al., 2006), is a metaheuristic method centered on the idea of Darwin's theory of evolution. This algorithm is motivated by biological operators (mutation, crossover and selection) for the production of quality solutions towards solving optimization and search problems. G.A process begins with the population of randomly generated candidates (phenotypes) with a highly distinguishable set of properties known as chromosomes which can be mutated or altered. Each phenotype is usually represented as binary strings of 0s and 1s regarded as the first generation, but other encodings are possible as well (Cha at al., 2009).

The initial generation undergoes an iterative process where the fitness of each candidate in the generation is evaluated. The candidate fitness is further determined by the objective function(s) provided for the problem under optimization. Fitted candidates are stochastically selected with genes modified to form a new generation which is used in the next iteration until a satisfactory fitness level is reached or number. Genetic representation of the solution domain and formulation of fitness function for its evaluation are the two major conditions required in the implementation of the GA (Doval, Mancoridis & Mitchell, 1999). Once these are clearly defined, GA proceeds to initialize a population of solutions, improves it through repetitive application of the mutation, crossover, inversion and selection operators. The stages are further summarized in the sub-sections below.

a. Initialization

The initialization output is a collection of randomly generated candidates possessing a wide range of possible solutions for the surveyed optimization problem. Additionally, the population size itself is closely related to the nature of the problem being solved and the availability of candidates for the solution. The resulting solutions, as is often the case, are concealed in areas where optimal solutions are likely to be found (Challagulla et al., 2008).

Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

b. Selection

The selection of candidates is determined by the fitness function with the main aim of rating the fitness value of each solution and preferentially selecting the best solutions after each successful iteration (Jaspree, 2011). Candidates are represented in an array of bits with values such as zero and one. Although fitness function is always problem-dependent, it also measures the quality of the represented solution. Defining fitness functions may be quite hard in some problems (Singh & Chug, 2017), but the use of simulation tools or interactive genetic algorithms will be of great assistance.

c. Genetic mutation and operations

For a set of new candidates (offsprings) marking the second generation to be created, there must be an interaction between the parents. Similarly, for each new solution, a combination of genetic operations in terms of cross-over and mutation is performed on the initial candidates (parents) selected from the pool of generated solutions (Haznedar & Kalinli, 2016). The new solution (offsprings) typically shares many characteristics with the parents, thus, making them more suitable since only the best organisms from the first and subsequent generations are selected for breeding to yield the best optimal solution. New parents are also selected for each new child and the process continues until a new population of solutions of appropriate size with increased state of fitness for the population is generated. Other heuristics can be adopted for a faster and more robust operation (Cha & Tappert, 2009)

d. Termination conditions.

Termination conditions are specified for the successive generations of offspring. Possible conditions for the termination include one or more of these factors:

i. Deriving optimal solution satisfying the given conditions
ii. Reaching the fixed number of generations
iii. Keeping within the allocated budgets (memory and computational time)
iv. Manual inspection and reaching a point where no better results could be generated

2.1 Multi Objective Optimization Algorithms

The process of selecting the best elements (with consideration to certain criteria) from given alternatives is termed optimization. More generally, it involves finding "best available" values attributes which satisfy some objective functions for a given problem (Zavala et al., 2014). It is a commonly used term not only in computer science, mathematics and engineering but also in our daily life ranging from budgeting our finances, planning schedules or trips to determining the number of items.

The task of determining a single objective function for determining the best solution to an optimization problem is known as single-objective optimization (Suman, 2004) but when two or more functions are involved in seeking one or more optimum solutions to a problem, such is regarded as a multi-objective optimization problem. Naturally, a trade-off usually exists between different goals in real life as there is no singular optimal approach to multi-objective optimization problems (Jaddan et al., 2008). So, when more than one objective function is involved in an optimization problem, the task of seeking one or more optimal solutions is known as multi-objective optimization.

Commonly used approaches in multi-objective algorithms are such that the objective functions are merged using a weighted sum method (scalarizing) to generate a single composite function. The Pareto optimal solution seems ideal for this approach but with a limitation for not being able to generate all accurate points to a scalarized problem. Secondly, all but one target could be moved to a constrained condition with the aim of getting a set of Pareto Optimal points (Chen & Guestrin, 2016) with the same process repeated for all combinations of values. Another approach is to select the entire optimal solution set at once, which may be quite challenging to achieve.

Better results are obtained with heuristic algorithms due to their possession of higher-level techniques or methods aimed at finding precise solutions to complex optimization problems with no available precise methods(Zavala et al., 2014). Continuous increase in the acceptance rate of heuristic algorithms is

Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

due to their adaptability for use in both combinatorial and continuous functions optimization problems and their ability to converge multiple generated solutions to an optimal solution at a single run (Suman 2004).

2.2 Related works

Several software metrics are used in determining the quality of softwares. In research performed by Okutan and Yıldız (2014) on various machine learning techniques, they proved that machine learning provides capabilities for software defect prediction. Their studies significantly helped developers in the usage of useful software metrics and suitable data mining techniques towards enhancing the quality of softwares. The most effective metrics identified in their research for software defect prediction are Response for class (ROC), Line of code (LOC) and Lack of Coding Quality (LOCQ). Singh and Chug (2017) also showed that other factors including Coupling Between Objects (CBO), WMC, Line of code (LOC), and RFC are effective in predicting softwares defects. Malhotra and Singh (2011) also revealed that AUC is an effective metric that could be used to predict faulty modules in the early phases of software development and to improve the accuracy of Machine Learning techniques.

Most software programs contain syntactic structures and semantics information which could also be used in the prediction of defective software programs. It is this basic idea and knowledge that were leveraged with deep learning technique in the development of software defect prediction framework via an attention-based recurrent neural network (DP-ARN N) by Yamaguchi et al., 2012. Exploring deep programs semantics. Peng et al. (2008) also deployed an ensemble method comprising of Bagging, Boosting and Stacking Based techniques for 10 publicly NASA MDP dataset to assess the quality of ensemble approaches in software fault prediction with the analytical hierarchal process. Based on the performance measure AdaBoost with a decision tree generated the best result accuracy of 92.53 %.

Traditional defect prediction focuses on designing discriminative artificial metrics to achieve higher model accuracy. Halstead features (1977) depend on the number of operators and operands, CK features (Jureczko & Spinellis, 2010) based on object-oriented programs and dependency-based McCabe features are the divisions of manual metrics adopted traditionally. Detecting defective software by relying on static code isn't a very good approach because both buggy and clean codes snippets may have the same number of static code attributes making it complex to differentiate (Costa et al., 2007).

The fault proneness of object-oriented programming via k-mean based clustering approach was performed by Jaspree (2011). Used, for the purposes of their investigation, was a model built on the basis of clustering algorithms (EM and X-means) from three promise repository data (AR3, AR4, AR5) to predict software faults. At first, normalization was performed on the datasets. The CfsSubsetEval attribute selection was compared to the non-attribute reduction dataset used. The experiment result showed that X-means have more accuracy (90.48%) than other models for AR3 without attribute reduction. Saiqa et al. (2015) combined multiple data sets including AR1, AR6, CM1, KC1, and KC3 with various machine learning methods for software bugs prediction. Performance measures of each dataset with methods were analyzed and the conclusion reached was that Support Vector Machine, Multilinear Programming and bagging had high accuracy and performances.

Some of the most popular data mining techniques (k-Nearest Neighbors, Naïve Bayes, C-4.5 and Decision trees) were analyzed and compared. Their advantages and disadvantages were also highlighted. The results of the study showed that different factors were affecting the accuracy of each technique such as the nature of the problem, the used dataset and its performance matrix. In Sharma and Chandra (2018) analysis of the applicability of various Machine Learning methods for fault prediction, they also added to their study the most important previous researches about each ML technique and the current trends in software bug prediction using machine learning.

3 Model Design and Methodology

The research methodology adopted involves implementing and evaluating a multi-objective optimization model for predicting bugs in software using genetic algorithm, testing statistical validity and significance of the generated model. The model was developed using the National Aeronautics and Space Administration (NASA) dataset comprising of fourteen similar sub-datasets (including CM1,

Technical University of Varna
Annual Journal
ISSN 2603-316X (Online)
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11

JM1, KC1, KC2, KC3, MC1.MC2, MW1, MW2, PC1, PC2, PC3, PC4 and PC5) all in CSV format. This data has been used by many researchers for the prediction and classification of software bugs. Furthermore, the choice for this data is its wide inclusion of salient and relevant software attributes for determining the effectiveness of software.

a. Dataset description

A publicly available NASA dataset was used due to its composition of well classified salient features (identical, constant, missing value, conflicting value, implausible values and total problems). The classification instance by features shown in Table 1. helps to identify the most distinguishing characteristics with the most prominent attributes

Table 1. Detailed data quality analysis of the NASA defect data sets by features

	A		B		C		D		E		F	
	Identical features		Constant features		Features with missing values		Features with conflicting values		Features with implausible values		Total problem features	
Data Set	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom
CM1	2	0	3	0	1	0	2	14	0	6	6	15
JM1	0	0	0	0	0	5	9	15	0	6	9	16
KC1	0	0	0	0	0	0	4	15	0	6	4	16
KC2	n.a.	0	n.a.	0	n.a.	0	n.a.	14	n.a.	6	n.a.	15
KC3	0	0	1	0	1	0	0	0	1	1	3	1
KC4	27	n.a.	26	n.a.	0	n.a.	3	n.a.	0	n.a.	30	n.a.
MC1	0	0	1	0	0	0	3	3	1	1	5	4
MC2	0	0	1	0	1	0	0	0	0	0	2	0
MW1	2	0	3	0	1	0	0	0	0	0	4	0
PC1	2	0	3	0	1	0	4	14	1	6	8	15
PC2	3	0	4	0	1	0	2	2	1	1	8	3
PC3	2	0	3	0	1	0	2	2	1	1	7	3
PC4	2	0	3	0	0	0	7	7	1	1	11	8
PC5	0	0	1	0	0	0	3	3	1	1	5	4

3.1 Research Hypothesis

These research hypotheses are formulated and tested on a statistical T-test scale to initially determine the correlating effects of the parametric features and the number of code lines on software efficiency.

a. Number of Parametric Features (F):

Ho_1: There is no significant relationship between the number of parametric features and overall software efficiency.

Ha_1: There is a significant relationship between the number of parametric features and overall software efficiency.

Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

b. Number of Lines (L):

Ho_2: There is no significant relationship between the number of lines and overall software efficiency

Ha_2: There is a significant relationship between the number of lines and overall software efficiency

3.2 Research Methodology

The most significant parametric features representing the entire dataset are renamed as F1, F2, F3, F4, F5 and F6 to serve as the first set of input attributes. Consequently, the second input variables dependent on the number of lines are classified as N1, N2, N3, N4, N5 and N6 as shown in Table 2.

Table 2. The G.A-based software bugs predictor variable specification

S/n	Parametric Input	S/n	Number of lines
F1	HALSTEAD LEVEL: Mental effort for software design	N1.	LOC_BLANK: Blank Lines of code available in a program
F2.	NODE COUNT: available number of data structures used during software development	N2.	PARAMETER COUNT: Number of variables for passing information between functions
F3.	CYCLOMATIC COMPLEXITY: Number of decision counts in a software	N3.	OPERATOR NUMBERS : Number of operators in the software
F4.	MULTIPLE CONDITION COUNT: Number of count conditions	N4.	NUMBER OF OPERANDS: Count of instructions specifying data to be operated on
F5.	HALSTEAD VOLUME: Number of unique operators and operand occurrence	N5.	COMMENT PERCENT: Percentage number of annotations in the source code
F6.	DESIGN COMPLEXITY :	N6.	LOC CODE AND COMMENT: Available lines of code and comments in a program.

The optimization functions for these input variables are represented mathematically in equations. (1) and (2) respectively.

Minimize Objective function: Parametric Features

$$Z_1 = 0.030 + 0.036(F1) + 0.040(F2) + 0.060(F3) + 0.00(F4) + 0.014(F5) + 0.008\ (F6) \quad (1)$$

Minimize Objective function: Number of lines

$$Z_2 = 14.839 + 76.461(N1) + 32.254\ (N2) + 13.692\ (N3) + 18.879\ (N4) + 2.808\ (N5) - 0.034(N6) \quad (2)$$

Furthermore, to determine the validity of the factors selected from the statistical T-test model, we also adopted a computational Principal Components Analysis (PCA) algorithm as shown in Fig.2 to perform a comparative analysis. A Genetic Algorithm-based multi-objective optimization model was later developed by applying the algorithm as illustrated by the flowchart in Fig. 1 on the input variables.

Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

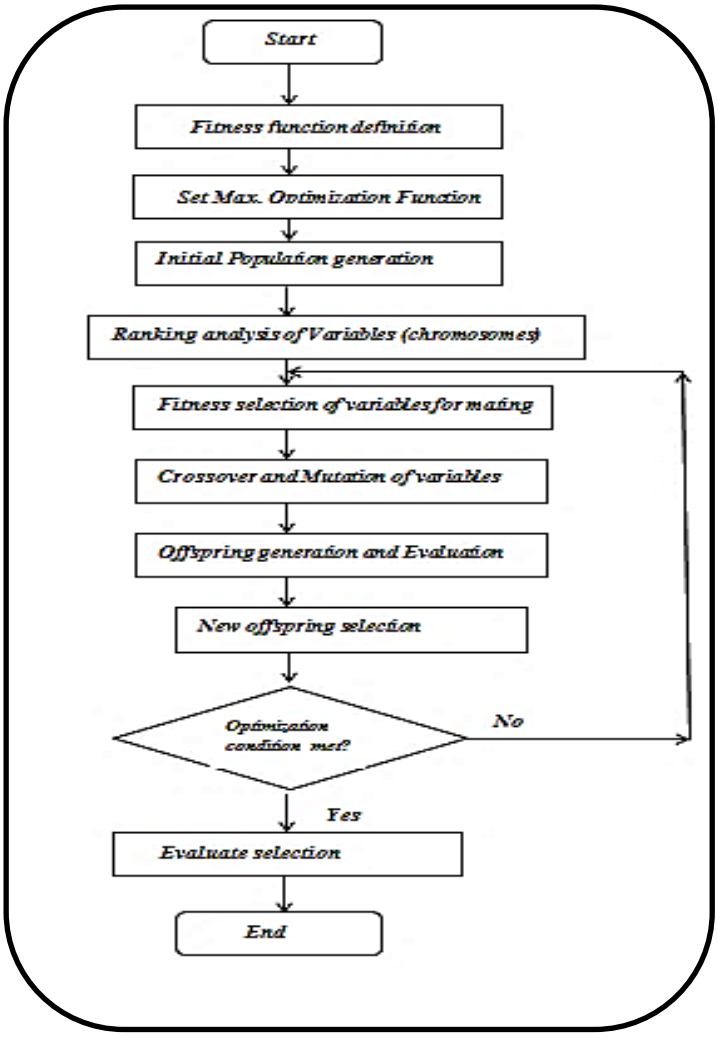


Fig. 1. Flowchart of the GA-based multi-objective optimization model

4 Implementation

The implementation of the developed GA-based multi-objective optimization model was performed on MATLAB 9.5 version installed on an Intel Core I5 PC with1TB HDD and 4GB RAM. A multi-objective fitness function is generated and mathematically represented by equ. (3) and (4), respectively.

Function y = EVA_multiobjective (α, β)

$$Y = y(1) = 0.030 - 0.036*(\alpha_1) + 0.040*(\alpha_2) + 0.060*(\alpha_3) + 0.00*(\alpha_4) + 0.014*(\alpha_5) + 0.008*(\alpha_6) \quad (3)$$

$$y(2) = 14.839 + 76.461*(\beta_1) + 32.254*(\beta_2) + 13.692*(\beta_3) + 18.879*(\beta_4) + 2.808*(\beta_5) - 0.034*(\beta_6) \quad (4)$$

where α = the first set of attributes for the parametric features;
β = the second set of input variables dependent on the number of lines.

Other parameters for the multi-objective genetic algorithm are set as follows:

- Solver : gamultiobj-Multiobjective optimization using Genetic Algorithm
- Number of variables: 6
- Lower bounds=10, 20, 30, 40, 50, 60
- Upper bounds=20, 30, 40, 50, 60, 70
- Population type: Double vector

Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

- Creation function: Constrain dependent
- Selection: Tournament selection with size=2
- Reproduction: Crossover fraction = 0.8
- Mutation function: Constraint dependent
- Crossover: Crossover function: Intermediate=1.0
- Migration: Direction (both) with fraction of 0.2 and interval of 20
- Distance measure function: distance crowding
- Plot functions = Average Pareto distance, Rank histogram, Pareto front, Average Pareto spread.

4.1 Algorithms: Dimensionality Reduction using Principal Component Analysis (PCA).

Major operations performed by this model are depicted by different algorithms, commencing with the feature reduction algorithm represented in Fig.2. Fig.3 depicts the initial population generation process, Figs. 4 and 5 are the crossover and mutation algorithms also deployed in the optimization process to select features with improved attributes to help detect buggy software with limited lines of codes and features.

INPUT: Raw NASA DataSet ; Ds1, ..., DS13 n n-dimension vector, n
OUTPUT: Cleaned NASA Data R1, ..., Rk k -dimension vector, R з k≤ n Э

Process:
1.BEGIN
2 .{
3 DS ← n x k data matrix with α in each row
4 DS ← $\frac{1}{n}\sum_{i=1}^{n} DSi$
5 -DSi in DS // from each row
6 COV ← $\frac{1}{n-1}$ DS1 x DSn compute eigenvalue e1, ..., en of COV and sort them
7 Compute matrix w which satisfies w-1 x COV X w = d// d representing the diagonal matrix of ei genvalue of COV
8 Return k- dimension// the first k column of V
9 }
10 END

Fig. 2. Reduced feature selection algorithm

Input: Prediction features (F), Number of lines (N)
Output: Fittest variables (v) , v1, v2

Process:
1. ***Begin***
2. ***Generate*** *random population a, b from F and N*
3. Y ← a x b
4. ***While*** *Y is not empty* ***do***
5. T1 ← Fit-featureSelection1
6. T2 ← Fit-featureSelection2
7. Select a random number ***r*** *з 0 ≥ r < 1*
8. ***If*** *(r > T)* ***do*** *// if r is less than the crossover rate*
9. crossover(T1, T2)
10. ***else*** *return new****offspring 1*** *← T1 , new****offspring 2*** *← T2*
11. mutation(new offspring1, newoffspring2)
12. V ← ***offspring 1****,* ***offspring 2***
13. Return v
14. ***End if***
15. ***End while***
16. ***End***

Fig. 3. Genetic Algorithm Process

Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

Input: m, n (parent 1, 2),crossover rate,r.
Output: t1, t2 (offsprings)

Process:
1. m1← m and n1← n
2. ***while*** *crossover rate r ← true do:*
i. Task J, a randomly selected task acts as the crossover point of the offsprings.
ii. create a swap point (p1) for necessary operations for crossover
iii. Randomly select a swap point (p2) for swapping
iv. Collect similar chromosomes in Set1(J)and subsets in (J) in locations zero(0) and p1
i. swap chromosomes (u)equals subsets in (J) with equal positions or greater
vi. ***Repeat*** *steps iv and v*
vii. ***FOR*** *every subset S2 in new swapSet2(J)* ***do****:*
(a) Add S2 as a new subset in remainingSet1(J).
(b) Join S2 with an existing subset (X1) in remainingSet1 (J).
(c) Select a subset X1 in remainingSet1(J), eliminate X1 elements found in S2 and add S2 to remaining.
viii. ***Repeat*** *Step vii*
i. D1(J)← remainingSet1(J) and D2(J)← remainingSet2(t).
j. ***Repeat*** *steps ii. to viii*
k. Update the related tasks to J.
3. ***Return*** *t1 and t2.*

Fig. 4. Crossover operation algorithm

Input: t1and t2
Output: Mutated individuals

Process:
1: ***While*** *Mutation rate r ≠ 0* ***do***
2. ***for every*** *task (J) involving t1 and t2 with condition function f(J)*
3. ***Do*** *any of the following:*
a. Select a subset X in f(J)and add a task J1 to X, where J1 belongs to the set of tasks in the individual.
b. Select a subset X in f(J)and remove a task J1 from X, where J1 belongs to X. If X is empty after J1 removal, exclude X from f(J).
c. Redistribute the elements in f(J).
3. ***Repeat*** *2 but use the condition (J) instead of F(J).*
4. Return Mutated individuals

Fig. 5. Mutation algorithm

Input: NASA Dataset

Output: Optimized Dataset with Improved Population

Process:

1. *Apply Dimensionality reduction using PCA*
2. *Generate multi-linear regression equations*
3. *Select a random initial population P_t (S)*
4. *While (Stopping Criteria are NOT met)* ***do***
5. *Perform Genetic Algorithm optimization*
6. *Evaluate fitness of the population*
7. *Select parents using a stochastic selection method*
8. *Apply crossover and mutation*
9. *Update P_t, ND and set t=t+1// processed dataset*
10. *End while*
11. *Return ND and P_t*
12. *End.*

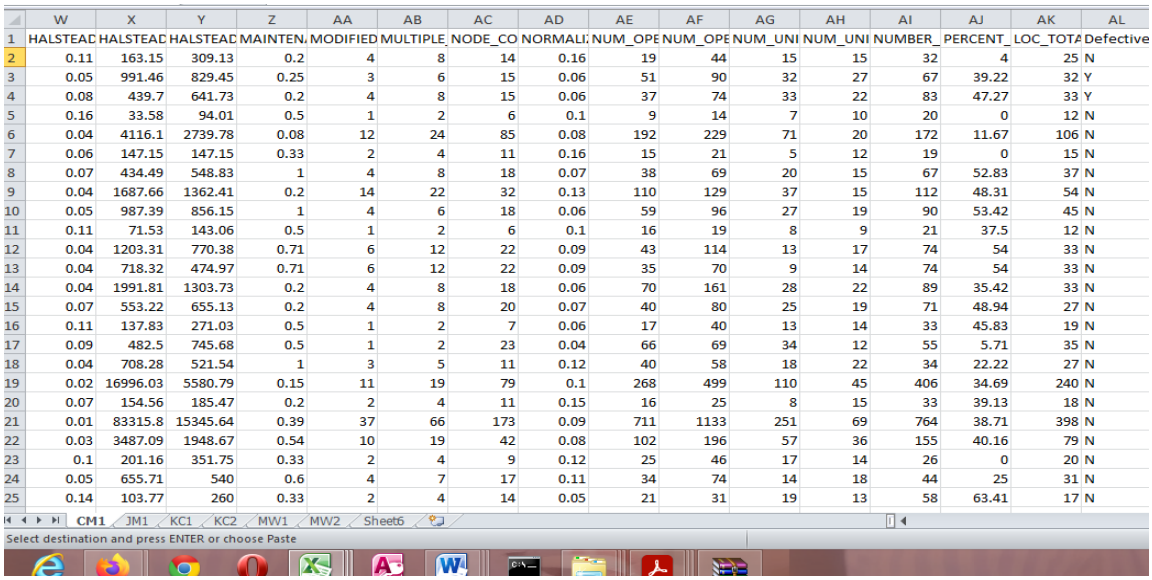
Fig. 6. Algorithm of the proposed GA based optimization model

Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

4.2 Implementation results and discussion

An instance of the collated NASA dataset containing thirty-eight (38) valued attributes for buggy software detection is shown in Fig. 7. The system's memory space is efficiently utilized, and redundancy of data is eliminated through a generated component transformed matrix derived by applying Varimax and Kaiser normalization technique. Six major components as displayed in **Table 3** after normalization are selected and used for developing the model.

For further implementation and accuracy of the model, the reduced dataset was subjected to genetic algorithm (GA) optimization with the MATLAB software. In Fig.8, an interface depicting the written objective functions in MATLAB scripts for fitness selection of chromosomes and offsprings for a better and optimized result was also specified.



	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL
1	HALSTEAD	HALSTEAD	HALSTEAD	MAINTEN	MODIFIED	MULTIPLE	NODE_CO	NORMALI	NUM_OPE	NUM_OPE	NUM_UNI	NUM_UNI	NUMBER_	PERCENT_	LOC_TOTA	Defective
2	0.11	163.15	309.13	0.2	4	8	14	0.16	19	44	15	15	32	4	25	N
3	0.05	991.46	829.45	0.25	3	6	15	0.06	51	90	32	27	67	39.22	32	Y
4	0.08	439.7	641.73	0.2	4	8	15	0.06	37	74	33	22	83	47.27	33	Y
5	0.16	33.58	94.01	0.5	1	2	6	0.1	9	14	7	10	20	0	12	N
6	0.04	4116.1	2739.78	0.08	12	24	85	0.08	192	229	71	20	172	11.67	106	N
7	0.06	147.15	147.15	0.33	2	4	11	0.16	15	21	5	12	19	0	15	N
8	0.07	434.49	548.83	1	4	8	18	0.07	38	69	20	15	67	52.83	37	N
9	0.04	1687.66	1362.41	0.2	14	22	32	0.13	110	129	37	15	112	48.31	54	N
10	0.05	987.39	856.15	1	4	6	18	0.06	59	96	27	19	90	53.42	45	N
11	0.11	71.53	143.06	0.5	1	2	6	0.1	16	19	8	9	21	37.5	12	N
12	0.04	1203.31	770.38	0.71	6	12	22	0.09	43	114	13	17	74	54	33	N
13	0.04	718.32	474.97	0.71	6	12	22	0.09	35	70	9	14	74	54	33	N
14	0.04	1991.81	1303.73	0.2	4	8	18	0.06	70	161	28	22	89	35.42	33	N
15	0.07	553.22	655.13	0.2	4	8	20	0.07	40	80	25	19	71	48.94	27	N
16	0.11	137.83	271.03	0.5	1	2	7	0.06	17	40	13	14	33	45.83	19	N
17	0.09	482.5	745.68	0.5	1	2	23	0.04	66	69	34	12	55	5.71	35	N
18	0.04	708.28	521.54	1	3	5	11	0.12	40	58	18	22	34	22.22	27	N
19	0.02	16996.03	5580.79	0.15	11	19	79	0.1	268	499	110	45	406	34.69	240	N
20	0.07	154.56	185.47	0.2	2	4	11	0.15	16	25	8	15	33	39.13	18	N
21	0.01	83315.8	15345.64	0.39	37	66	173	0.09	711	1133	251	69	764	38.71	398	N
22	0.03	3487.09	1948.67	0.54	10	19	42	0.08	102	196	57	36	155	40.16	79	N
23	0.1	201.16	351.75	0.33	2	4	9	0.12	25	46	17	14	26	0	20	N
24	0.05	655.71	540	0.6	4	7	17	0.11	34	74	14	18	44	25	31	N
25	0.14	103.77	260	0.33	2	4	14	0.05	21	31	19	13	58	63.41	17	N

CM1 JM1 KC1 KC2 MW1 MW2 Sheet6

Select destination and press ENTER or choose Paste

Fig. 7. Instance of CM1 dataset contained in the NASA database

Table 3. Component Transformation Matrix

Principal Component Analysis and Varimax with Kaiser Normalization						
Components	1	2	3	4	5	6
1	0.673	0.636	0.25	0.276	0.055	0.029
2	0.501	-0.561	-0.427	0.485	-0.104	-0.082
3	-0.045	0.455	-0.709	-0.184	-0.468	-0.187
4	-0.024	0.15	-0.489	-0.027	0.848	0.133
5	0.538	-0.227	0.015	-0.807	0.049	-0.07
6	0.056	-0.015	-0.112	-0.057	-0.214	0.967

Technical University of Varna
Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

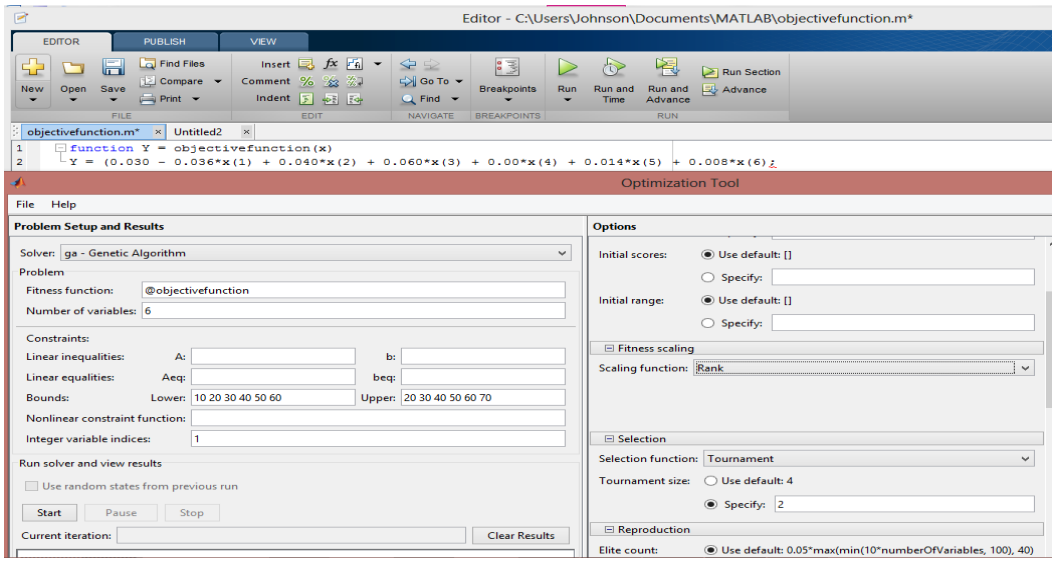


Fig. 8.G.A optimization interface for fitness selection with specification

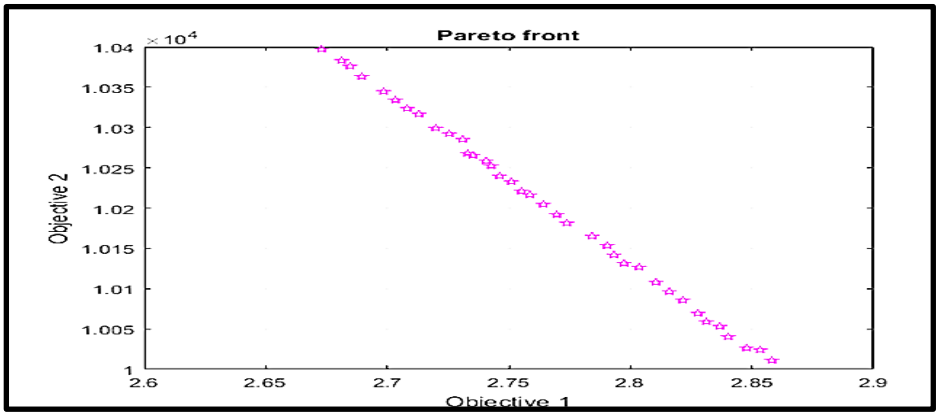


Fig. 9. G.A-based optimization objective functions plot.

Sample plots derived from the optimization procedures and fitness selections of offsprings are depicted in Figs. 10, 11 and 12. Fig. 10 is a plot showing the distance of individuals at each iteration. Ranks and spread of variables are also depicted in Figs. 11 and 12, respectively.

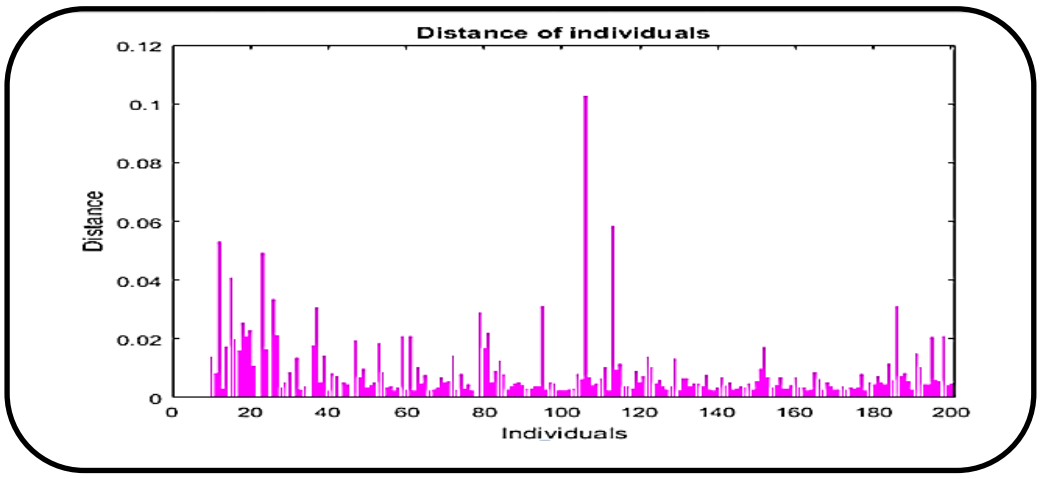


Fig. 10. Distance of individuals at successive iteration

Technical University of Varna
Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

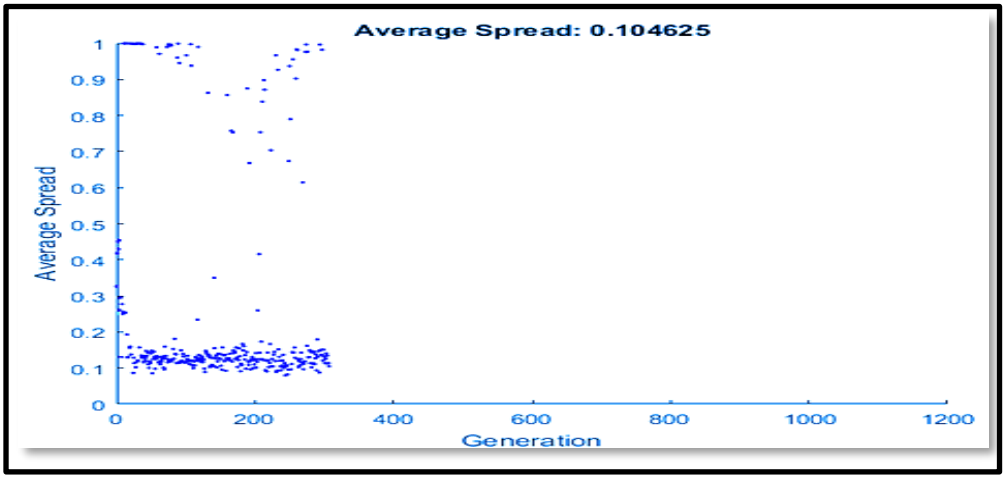


Fig. 11. Average spread of variable features

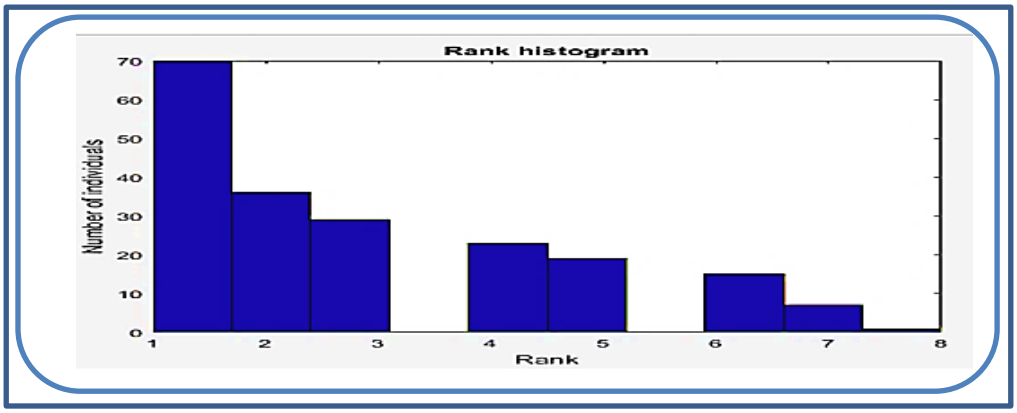


Fig. 12. Ranks of variable features

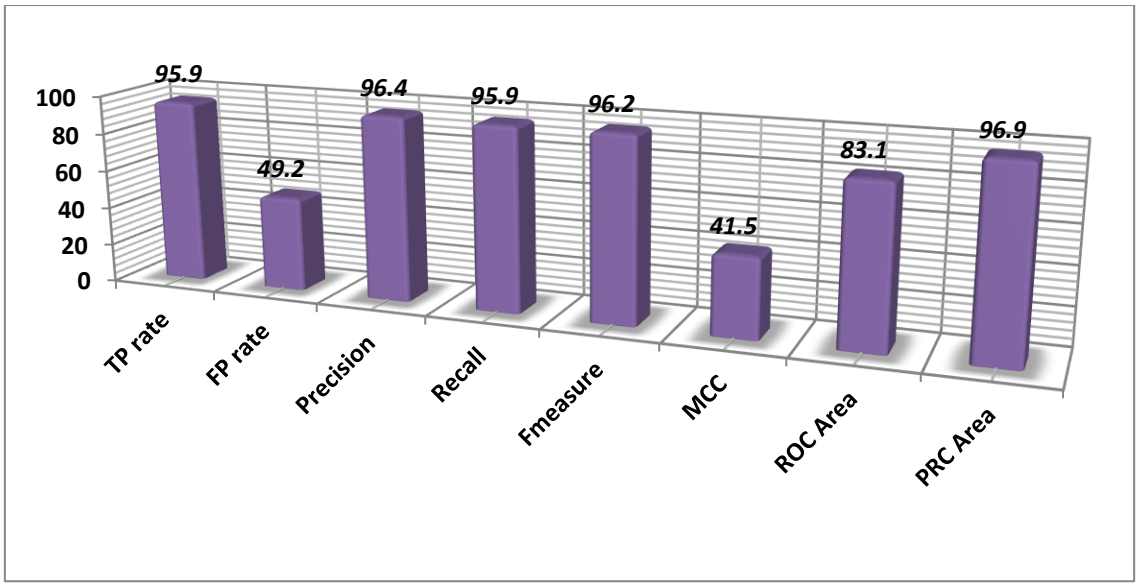


Fig. 13. G.A based multi-objective optimization model Accuracy metric

Technical University of Varna Annual Journal
https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

5 Conclusion

Proposed, in the present paper, is a genetic algorithm-based multi-objective model to predict software bugs. Multi-objective optimization has proven to be successful in various fields of machine learning. Its usage for predicting software bugs is one of its numerous applications as software is very essential to the functionality of any operational hardware.

The adoption of PCA for feature selection proved conclusively that the six significant software features are sufficiently able to map the entire wide range data obtained from the publicly available NASA dataset for software validity check. Thus, reducing computational time and memory usage led to the normalization of data. Comparison of the statistical technique (t-test) generated results with the computational PCA on dataset returned closely related range of values.

Furthermore, the results generated by the Genetic Algorithm-based multi-objective optimization software bugs predictor have greatly depicted the accuracy of this model. Individual response characteristics are obtained at optimal sets of levels for the processed features derived from the statistical response surface techniques. The results obtained are within 95% to 97% prediction accuracy of the respective response intervals acquired through the optimized model completed in a MATLAB-based software environment. This also indicates that the optimal values are within the specified range of processed variables for accurate software bugs prediction.

Future research on the prediction of software bugs will explore the practical application of Differential Evolution (DE), ensemble machine learning algorithms and other computational meta-heuristics algorithms for software predictions.

References

Akmel, F., Birihanu, E. Siraj, B. (2017). A literature review study of software defect prediction using machine learning techniques. *Int. J. Emerg. Res. Manag. Technology*, *6*(6), 300-306. https://doi.org/10.23956/ijermt.v6i6.286

Bavisi, S., Mehta, J., & Lopes, L. (2014). A comparative study of different data mining algorithms. *International Journal of Current Engineering and Technology*, *4*(5), 3248-3252.

Catal C., Diri, B. (2009). A systematic review of software fault prediction studies. *Expert Systems with Applications, 36* (4), 7346–7354. https://doi.org/10.1016/j.eswa.2008.10.027

Cha, S. H., & Tappert, C. C. (2009). A genetic algorithm for constructing compact binary decision trees. *Journal of pattern recognition research*, *4*(1), 1-13. https://doi.org/10.13176/11.44

Challagulla, V. U. B., Bastani, F. B., Yen, I. L., & Paul, R. A. (2008). Empirical assessment of machine learning based software defect prediction techniques. *International Journal on Artificial Intelligence Tools*, *17*(02), 389-400. https://doi.org/10.1142/S0218213008003947

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD, International conference on knowledge discovery and data mining*, (22), 785-794. https://doi.org/10.1145/2939672.2939785

Costa, E. O., de Souza, G. A., Pozo, A. T. R., & Vergilio S. R. (2007). Exploring Genetic Programming and Boosting Techniques to Model Software Reliability. *IEEE Transactions on Reliability, (*56): 422-434. https://doi.org/10.1109/TR.2007.903269

Doval, D., Mancoridis, S. & Mitchell, B. S. (1999). Automatic clustering of software systems using a genetic algorithm. *Proc. Conference of Software Technology and Engineering Practice*, England, 73-81. https://doi.org/10.1109/STEP.1999.798481

https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

Gupta, Dharmendra, L. & Kavita S. (2017). Software bug prediction using object-oriented metrics. *Sādhanā (1),* 1-15.

Haznedar, B. & Kalinli, A. (2016). Training ANFIS Using Genetic Algorithm for Dynamic Systems Identification. *Int j Intell Sys appl eng*, *4*(1), 44-47. https://doi.org/10.18201/ijisae.266053

Jaspree, K. (2011). A k-means Based Approach for Prediction of Level of Severity of Faults in Software System. In *Proceedings of International conference on Intelligent Computational Systems*.

Jureczko, M., & Spinellis, D. (2010). Using object-oriented design metrics to predict software defects. *Models and Methods of System Dependability*. *Oficyna Wydawnicza Politechniki Wrocławskiej*, 69-81.

Kim, S., Zhang, H., Wu, R. and Gong, L. (2011). Dealing with noise in defect prediction. *Proceeding of the 33rd International Conference on Software Engineering, ICSE* (11), 481–490. https://doi.org/10.1145/1985793.1985859

Malhotra, R. & Singh, Y. (2011). On the applicability of machine learning techniques for object-oriented software fault prediction. *Software Engineering: An International Journal 1*(1), 24-37.

Naidu, M. S., & Geethanjali, N. (2013). Classification of defects in software using decision tree algorithm. *International Journal of Engineering Science and Technology*, *5*(6), 1332-1342.

Okutan, A., & Yıldız, O. T. (2014). Software defect prediction using Bayesian networks. *Empirical Software Engineering 19*(1),154-181. https://doi.org/10.1007/s10664-012-9218-8

Peng, Y., Kou, G., Wang, G., Wu, W., & Shi, Y. (2011). Ensemble of software defect predictors: an AHP-based evaluation method. *International Journal of Information Technology & Decision Making*, *10*(01), 187-206. https://doi.org/10.1142/S0219622011004282

Rahman, F., Posnett, D. & Devanbu, P. (2012). Recalling the imprecision of cross-project defect prediction. *In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, (12)* 1–61:11 New York, NY, USA. ACM. https://doi.org/10.1145/2393596.2393669

Sharma, D. & Chandra, P. (2018) Software Fault Prediction Using Machine Learning Techniques. In *Smart Computing and Informatics* (pp.541-549). Springer, Singapore. https://doi.org/10.1007/978-981-10-5547-8_56

Shepperd, M., Bowes, D. & Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, *40*(6), 603-616. https://doi.org/10.1109/TSE.2014.2322358

Shivaji, S., Whitehead, E. J., Akella, R. & Kim, S. (2012). Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering*, *39*(4), 552-569. https://doi.org/10.1109/TSE.2012.43

Singh, P. & Chug, A. (2017). Software defect prediction analysis using machine learning algorithms. In *7th International Conference on Cloud Computing, Data Science & Engineering Confluence*, 212-232. IEEE. https://dpi.org/10.1109/CONFLUENCE.2017.7943255

Suman, B. (2004). Study of simulated annealing-based algorithms for multi-objective optimization of a constrained problem. *Comput. Chem. Eng. 28*(9) 1849–1871. https://doi.org/10.1016/j.compchemeng.2004.02.037

https://doi.org/10.29114/ajtuv.vol6.iss1.245
Vol.6 Issue 1 (2022)
Published: 2022-08-11
ISSN 2603-316X (Online)

Yamaguchi, F., Lottmann, M. & Rieck, K. (2012). Generalized vulnerability extrapolation using abstract syntax trees. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACM, (pp.359–368), Orlando, FL, USA. https://doi.org/10.1145/2420950.2421003

Zavala, G. R., Nebro, A. J., Luna, F., & Coello Coello, C. A. (2014). A survey of multi-objective metaheuristics applied to structural optimization. *Structural and Multidisciplinary Optimization*, *49*(4), 537-558. https://doi.org/10.1007/s00158-013-0996-4

Zhou, Y., Yang, Y., Lu, H., Chen, L., Li, Y., Zhao, Y., ... & Xu, B. (2018). How far we have progressed in the journey? an examination of cross-project defect prediction. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *27*(1), 1-51. https://doi.org/10.1145/3183339