Software Reliability Prediction using Correlation Constrained Multi-Objective Evolutionary Optimization Algorithm

Original Scientific Paper

Neha Yadav
KIET Group of Institutions, Delhi-NCR, Ghaziabad, India
nehayadav1508@gmail.com

Vibhash Yadav
Rajkiya Engineering College, Banda, India
vibhashds10@gmail.com

Abstract *– Software reliability frameworks are extremely effective for estimating the probability of software failure over time. Numerous approaches for predicting software dependability were presented, but neither of those has shown to be effective. Predicting the number of software faults throughout the research and testing phases is a serious problem. As there are several software metrics such as object-oriented design metrics, public and private attributes, methods, previous bug metrics, and software change metrics. Many researchers have identified and performed predictions of software reliability on these metrics. But none of them contributed to identifying relations among these metrics and exploring the most optimal metrics. Therefore, this paper proposed a correlation-constrained multi-objective evolutionary optimization algorithm (CCMOEO) for software reliability prediction. CCMOEO is an effective optimization approach for estimating the variables of popular growth models which consists of reliability. To obtain the highest classification effectiveness, the suggested CCMOEO approach overcomes modeling uncertainties by integrating various metrics with multiple objective functions. The hypothesized models were formulated using evaluation results on five distinct datasets in this research. The prediction was evaluated on seven different machine learning algorithms i.e., linear support vector machine (LSVM), radial support vector machine (RSVM), decision tree, random forest, gradient boosting, k-nearest neighbor, and linear regression. The result analysis shows that random forest achieved better performance.*

Keywords*: Reliability, Faults, Bugs, Object-oriented, Evolutionary optimization, Machine learning*

1. INTRODUCTION

Software development involves creating software with potential flaws, leading to negative consequences and financial losses [1]. To address these risks, decision-makers use software defect prediction (SDP) to anticipate faulty modules through testing and coding inspection [2]. Ensuring software reliability during development is challenging, especially with constant changes in the software engineering sector. Estimating models' accuracy can vary with different datasets, and improving reliability requires finding a suitable reliability allocation paradigm within constraints [3-6]. Software reliability refers to a system or component's probability of functioning properly in a specific environment for a certain period [7-10]. Evaluating software reliability during the design phase compares current reliability to previous performance, using models to analyze release time and estimate future reliability [11-17]. However, testing complex software becomes difficult, impacting the effectiveness of software models [18-24]. Various approaches, including machine learning techniques, have been explored to model software reliability and quality [25-27]. Throughout the software development lifecycle, estimation techniques are used in the early stages, while reliability growth models are used during testing to reduce failure rates and predict defect density after deployment. Inference techniques fit the curve to the data for software reliability forecasting and estimation, with failure intensity being a simpler measure often derived from the reliability estimate. Estimating software reliability is challenging due to unbalanced and inaccurate data. Researchers are exploring machine learning algorithms for software defect prediction but haven't delved into it extensively. Software reliability prediction aims to identify fault-prone components early, reducing costs and time while ensuring desired quality. Various prediction approaches for effort, privacy, quality, defect, cost, and reusability are still in the early stages of development [24]. Software Reliability Prediction (SRP) involves using machine learning to find problematic classes/modules before testing [6,25]. Neural networks and statistical approaches like logistic regression are used, but they lack optimal parameter selection for software fault determination. This paper proposes a novel multi-objective evolutionary-based strategy that integrates machine learning algorithms to predict software reliability by anticipating

errors during testing using past failure data [27]. Therefore, the major contribution of the paper is:

- The paper presented a novel approach using a machine learning algorithm for software reliability prediction using software metrics correlation-constrained multi-objective evolutionary optimization algorithm. The algorithm is based on the identification of the software metrics based on certain hypotheses.
- The paper presented the comparative performance evaluation of the presented hypothesis on different classification techniques.
- The comparative state-of-art is presented to show the effectiveness of the proposed method.

The rest of the paper is organized as section 2 presents the related contributions of researchers for software reliability prediction using optimization. Section 3 presented an overview of the proposed evolutionary optimization algorithm and a detailed flowchart and working of the proposed methodology along with a hypothesis description. Section 4 describes the result analysis and comparative analysis. Section 5 presents the discussion of the proposed method and the result obtained. Finally in section 6 conclusion and future scope are presented.

2. RELATED WORK

Dhavakumar and Gopalan [5] proposed a Chaotic-GWO. It uses the Pham-Zhang model to forecast parameters and improves reliability by approximating SRGM parameters using TEF and chaotic maps. The findings show a good convergence rate and a link between selected variables and the fitness criterion. The desired outcome is an automated SRGM using CGWO, eliminating the need for customer involvement. Rani and Mahapatra [22] presented the exponential software reliability model to quantify numerous aspects, particularly fault initiation and time-varying fault diagnosis frequency. To optimize software reliability while lowering allocation costs, an expanded particle swarm optimization (EPSO) is presented. Researchers do trials utilizing completely random testing-resource sets and the entropy function to modify performance. Gupta et al. [9] proposed a nonlinear MO-optimizer regarding data envelopment analysis (DEA) for choosing software systems in the context of optimum redundancy to assure software reliability. Kumar et al. [18] presented a novel comparison analysis to determine the most appropriate and accurate artificial neural network. In this study, we present a backpropagation-based feed-forward neural network for improving software reliability and accuracy. Jaiswal et al. [13] used machine learning techniques such as the adaptive neuro-fuzzy system (ANFIS), and other techniques are used to predict software reliability on a variety of datasets derived from the operating system. Sangeeta et al. [27] proposed a novel technique for optimizing parameter values depending on the Ant Colony Optimizer with differential evolutions idea of ecological space has boosted the exploration capabilities of the ABC algorithm. The suggested algorithm's efficiency is further validated by comparing it to hybrid PSO algorithms. The newly proposed ABCDE algorithm estimates the system's reliability to be up to 85 percent. Diwaker et al. [4] proposed a novel quantitative model is provided that uses series and parallel reliability frameworks to calculate the SR. To compare the best reliability value, the performance of the proposed method is assessed to the output of soft computing approaches PSO and Fuzzy logic. The development of a big combination of parameters increases the complexity of the suggested model as more components are included. Jabeen et al. [12] proposed a highly precise error iteration analysis technique (HPEIAM) based on error-residuals is suggested to improve the predictive performance of current PSRGMs. SRGMs compute residual errors repeatedly, improving and correcting prediction accuracy to the intended level. HPEIAM's performance is evaluated using various PSRGMs and two sets of actual software failure data, with three quality criteria in mind. Table 1 presents the comparative feature of related works with the presented work.

Table 1. Recent Contribution

Ref	Objectives	Contribution	Research Gaps
[6]	The aim is to cover all finite paths of the control flow graph of the software under the test.	Presents a memetic algorithm for automatically generating test data	To solve the software cost problem because Software testing is very time-consuming and expensive
[8]	To predict future software faults by deploying the classifier algorithms	Build the model and predict the occurrence of the software bugs based on historical data by deploying the classifiers Logistic regression, Naive Bayes, and Decision Tree	To solve Software bug prediction issues as bugs are a serious challenge for system consistency and efficiency
[11]	To estimate the software release time and cost of the testing effort	A high-precision error iterative analysis method (HPEIAM) Proposes to enhance the prediction accuracy	Limitations of them mean that their predictive capacities differ from one dataset to others
[12]	To predict software reliability and evaluate them based on selected performance criteria	Applied ML including (ANFIS), feed forward back propagation neural network, SVM, etc	To solve various challenges in developing highly reliable software.
[13]	The aim is to provide software reliability within the allotted time & budget.	Presents a genetic programming-based decision tree model which facilitates a multi-objective optimization in the context of the software quality classification problem	Improved by the allocated software quality-improvement resources, and on the project-specific costs of misclassifications.
[19]	Aims to remove the wrong solution during the algorithm execution process, and adds knowledge to improve the solution accuracy	Proposes parameter estimation method of software reliability model based on hybrid PSO-ABC	The existing software reliability models are nonlinear, and the parameter estimation of these models is difficult.

[21]	The aim is to provide an extensive comparison of the explanative and predictive power of well-known bug prediction approaches	Present a benchmark for defect prediction	Low Accuracy
[23]	Aim to predict bugs in software using machine learning	Used dynamic classifier for detection	Low Accuracy

3. METHODOLOGY

In this paper, we have designed an evolutionary algorithm-based software reliability prediction. For these object-oriented metrics are considered and some hypotheses are designed. These hypotheses are considered objective functions for evolutionary algorithms. The below sub-section describes the steps and working model of the proposed methodology.

3.1. MULTI-OBJECTIVE EVOLUTIONARY ALGORITHM

We apply evolutionary algorithms, which are a kind of artificial intelligence, in our suggested study. These algorithms optimize problems through mutation and recombination. Multi-objective evolutionary optimization improves machine learning performance. The algorithms simulate natural evolution, where solutions represent humans and the problem specification represents the environment. A population of solutions is created and evaluated using objective functions. Better solutions have a higher chance of being selected for recombination. Crossover and mutation are key reproduction phases. The population is regulated through replacement. The process is repeated until a stop criterion is met. These algorithms follow natural processes and focus on populations rather than individual solutions. The first generation is randomly initialized and evaluated. Traditional optimization approaches vary from evolutionary algorithms in the following ways:

- While some CCMOEO scans a population concurrently, some just look for a specific position.
- Just the objectives functional part and fitness level are required for CCMOEO, without derivation variables.
- It utilizes probability ideas.
- Because there were minimal restrictions on the definition of objective functional area, CCMOEO is often simple to use.

Natural selection for resources in the environment is what drives evolution. Those kids who are more likely to survive to live longer and pass on their genetic material. Asexual reproduction is used to encode genetic information, resulting in kids that are genetically identical to their parents. Nowadays, evolutionary algorithms may be found everywhere, having been effectively applied to a variety of issues in fields such as social systems, automated programming, signal processing, and bioinformatics. These methods are highly beneficial for optimizing outcomes from many domains [3]. There are several evolutionary algorithms strategies, such as ant colonies, bee colonies, and so on. We may utilize these types of algorithms to optimize the outcome in the area of software quality optimization. There are numerous stages to the optimization: i) An initialization of random solutions is formed, referred to as individuals. ii) The simulation model evaluates each individual's goal functions.

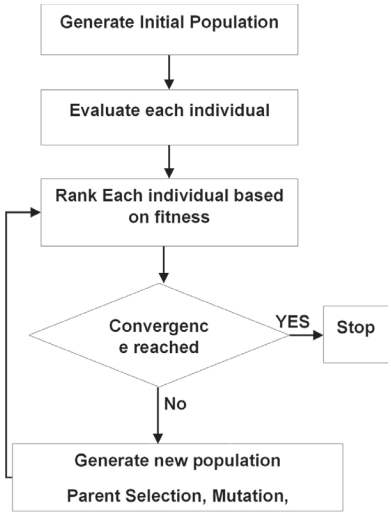


Fig. 1. Multi-Objective Evolutionary Algorithm

All software functionality, processing time, profiles, and costs are fixed input factors in our scenario since they do not vary throughout the optimization. iii) Each person is graded based on their "fitness," or the values of their goal functions. iv) Following the ranking of all individuals, the MOEA develops a new population of people (the next "generation") using the standard genetic algorithm operators of parent selection, crossover, and mutation. v) The MOEA generates a Pareto front after a certain number of generations (see Fig. 1).

3.2 OBJECT-ORIENTED DESIGN METRICS (OODMS)

One of the commonly used metrics in existing software reliability models is object-oriented design metrics (OODMs). Apart from this, there are several other Metrics also that can help determine software reliability growth. In the literature review, we have analyzed that there are several machine learning, optimization, and statistical techniques that are being used to predict software reliability. Many researchers also presented the correlation between OODMs and defective and non-defective classes of software. But relations among other metrics are still not that enlightening. For this, some hypotheses are presented in this paper which are discussed in the below sub-section. To prove this hypothesis, the paper proposed a software metrics correlation-constrained multi-objective evolutionary optimization algorithm whose steps are discussed below (Fig 2).

The methodology is divided into three basic steps: (a) pre-processing, (b) Correlation-Constraint Multi-objective Evolutionary Optimization, and (c) classification. These steps are described in detail below sub-sections.

3.2.1. Preprocessing

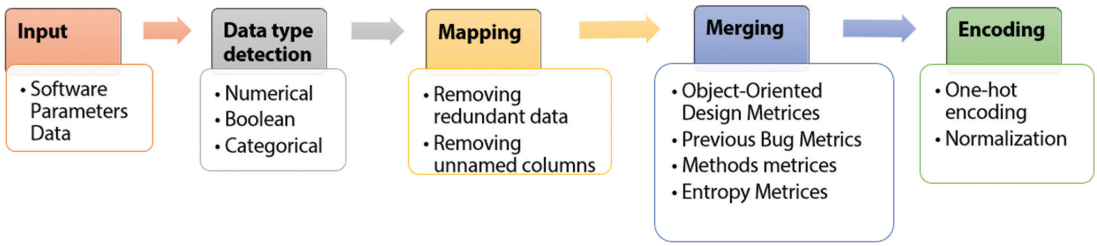
In this paper, the dataset contains a mixed set of data i.e., numerical and categorical parameters. Due to the dynamic nature of software reliability parameters, it is required to distinguish between parameters. For this pre-processing step is required. As machine learning can only handle numerical data. Therefore, for the proper estimation of the hyperparameter's performance, these mixed data are converted into an array of numerical data.

Transformation of Hyperparameters: In this step, a set of data features is used to estimate the reliability level of software. Fig 2 represents the pre-processing steps used for reliability prediction.

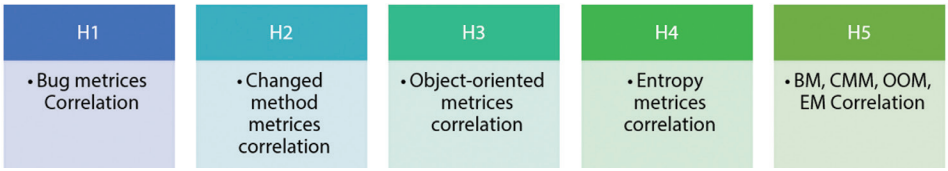
Data Type detection: In this step, a simple, type of data is detected i.e., numeric, categorical, and boolean. Some samples of datasets and their type are presented in Table 2.

Mapping: In this step, two major works are performed. In the first step, the redundant data are removed and in the second step, unnamed columns are removed. As it is known that data redundancy generally occurs when some parameters or Metrics are stored multiple times in the database. These redundant data don't seem a big issue but when the size of data increases, then these redundant data create unnecessary computational complexity for the learning model. Therefore, these data must be removed. Another major issue that occurs while the learning/analytics process is the presence of unnamed or unknown columns (software parameters) in the dataset. If these data are not removed then it will result in ambiguous analysis.

Merging: In this step, parameters related to software metrics are selected i.e., Object-Oriented Design Metrics, Previous Bug Metrics, Methods Metrics, and Entropy Metrics. These Metrics are most important for analytical purposes.



(a) Pre-processing



(b) Correlation-Constraint Multi-objective Evolutionary Optimization

(c) Classification

Fig. 2. Proposed Methodology

Table 2. Data Type Detection

Bug Metrices		Object-Oriented Metrics		Entropy Metrics	
Parameters	Type	Parameters	Type	Parameters	Type
class name	Mixed	CBO	Mixed	CvsEntropy	Mixed
Bugs Found	Integer	Fin	Decimal	CvsWEntropy	Decimal
Trivial bugs	Integer	Fout	Decimal	LinEntropy	Decimal
Major Bugs	Integer	No. of attributes	Decimal	LogEntropy	Decimal
Critical Bugs	Integer	No. of LOC	Decimal	ExpEntropy	Decimal
Priority Bugs	Integer	No of Methods	Decimal	Defect	Boolean
Defect	Boolean	Defect	Boolean	-	-

Encoding In this step, the one-hot encoding method is used to convert the merged data so that they can result in me better prediction results. so they can be provided to machine learning algorithms to improve predictions. This method is adopted because it of better prediction results of machine learning with those data that show no relationship to each other. After encoding data normalization is performed. Normalizing an attribute by evaluating the percentage of a value to the attribute's summation value is known as frequency normalization. It's described as:

$$x_i = \frac{x_i}{\sum_i x_i} \quad (1)$$

Frequency normalization also scales an attribute into [0,1].

3.2.2. Correlation-Constraint Multi-objective Evolutionary Optimization

In this section, an MOEA algorithm is proposed based on correlation constraints on the following hypothesis:

- Hypothesis 1 (H1): Previous bug metrics (BM) is unable to predict defect in the software.
- Hypothesis 2 (H2): Changed method metrics (CMM) is unable to predict defect in the software.
- Hypothesis 3 (H3): Object-oriented metrics (OOM) is unable to predict defect in the software.
- Hypothesis 4 (H4): Entropy metrics EM) is unable to predict defect in the software.
- Hypothesis 5 (H5): Bug matrices (BM), object-oriented metrics (OOM), changed method metrics CMM) and entropy metrics (EM) correlate them to predict defects in the software.

Scientific application is generally formulated as constrained optimization problems and mathematically it is represented as:

$$min\{f(X)\}=X=\{x_1, x_2,...x_n\} \; such \; that \; LB<x_n<UB$$
$$subjects \; to: \; g(X)\leq 0 \; \{inequality \; constraints\} \quad (2)$$
$$subjects \; to: \; h(X)=0 \; \{equality \; constraints\}$$

Where, $f(X)$= objective function,
$\{x_1, x_2,...x_n\}$ = decision vectors

LB and UB = *lower* and *upper bound* respectively

The equality constraint violation on all constraints is mathematically represented as:

$$G(X) = \sum_{i=1}^{n} G_i(X) \quad (3)$$

Where, i= inequality constraints

The solution that satisfies G(X) is termed a feasible solution. Therefore, the target of any constraint-oriented optimization is to locate optimal G(X). The evolutionary algorithm possesses a powerful searching ability that can solve above mentioned constraint problem efficiently. Therefore, this paper has adopted evolutionary optimization as a solution. The framework of the proposed correlation-constrained multi-objective evolutionary optimization algorithm (CCMOEO) is given in Fig. 3. Two basic steps are performed in this algorithm: learning and evolution. In the learning stage, five hypotheses are designed to establish a correlation between constraints and objective function. As a result, in the evolving stage, the correlation is used to direct the evolution process. To avoid a local optimum induced by complex constraints, correlation is employed to determine how much information of the objective function is utilized. As a result, the inhabitants can more easily enter the area. Constraint and goal function variation tendencies are more important to us in this work. That's why it's possible to catch a trend early on since the population is spread out throughout a large area. Learning is followed by evolution, which uses the rest of the computer's processing power.

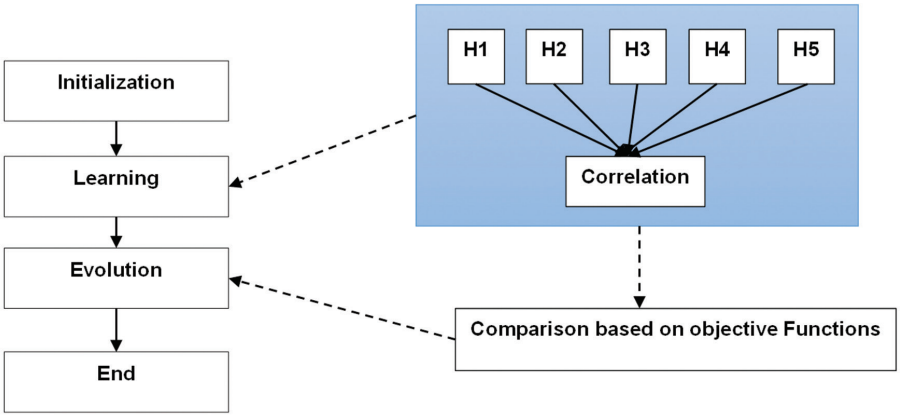


Fig. 3. CCMOEO Architecture

In the stage of learning, the notion of correlation index (CI) is proposed to mine the correlation between the variables. The closer the value of CI is to one, the greater the degree to which the constraints and the goal function are correlated. During the period of development, two different approaches are designed. The first method is referred to as the weighted sum updating strategy, and it states that the fitness value of an individual is determined by the weighted sum of normalized G(X) and normalized f.(X).

3.2.3. Classification

In this step, the optimized data is fed into a classifier for learning the pattern of failure and success of software metrics. For classification, the paper analyzed the performance of seven classifiers. These are discussed below:

Linear support vector machine (LSVM): A support vector machine is a machine learning model that is capable to simplify between two dissimilar classes if the fixed of categorized data is provided in the training set to the algorithm. The key purpose of the SVM is to distribute the data samples according to hyperplane and distinguish among different classes. Linearly Separable 2D Data is a two-dimensional database separated by line if we can distinguish positive from negative objects by a straight line. It does not matter if there is more than one such line. If data cannot be categorized, linearization cannot completely separate these two categories. For many non-linear databases, the line separator will still be "good enough" and segment multiple cases correctly.

Radial support vector machine (RSVM): The RBF kernel is one of the most popular kernels because it is the most general form of the kernel and resembles a Gaussian distribution. The RBF kernel function for two points Y_1 and Y_2 computes the similarity or how close they are to each other. An RBF kernel is a function whose value depends on the origin or distance from some point.

Decision tree: In machine learning, a decision tree is a predicting approach. It's a flowchart-like layout where every other block contains an attribute "test." Classification rules are represented by the results from one block to the next. Nodes may be classified into three categories: Squares are used to symbolize decision nodes. Ending nodes are depicted by triangles, whereas chance nodes are depicted by circles. The following is how a decision tree works: begin with the root node, which holds the whole dataset. By using Attribute Selection Measure, discover the perfect attribute in the dataset. Divide the root node into subgroups that include the best attribute's potential values. Create the node of the decision tree that holds the best attribute. Make new decision trees recursively using subsets of the dataset, Repeat this procedure till the nodes could no longer be classified and the last node is designated as a leaf node. It's utilized in data mining research. This is the most effective instrument for forecasting. The decision tree has a few benefits, like being easy to use and requiring minimal data preparation. The following are some disadvantages: it may result in too complicated trees, which is known as overfitting.

Random forest: It is a commonly used ML method that is classified as supervised learning. It may be used in ML for both classifiers and as well as for regression data. It is based on the notion of supervised methods, which is the act of combining numerous classifications to tackle a difficult problem and increase the individual's effectiveness. This approach consists of some decision trees, each of which can be built up of datasets retrieved from a training dataset, referred to as the bootstrap sample. The RF considers each tree's forecast and generates an outcome based on the plurality of predictions. The forest's enormous number of trees provides greater precision, avoiding the issue of overfitting.

Gradient boosting: Gradient boosting is an ML boosting technique. It is based on the assumption that combining the best subsequent modeling with the prior model reduces the total estimation error. To reduce mistakes, the key concept is to define target outcomes for the following concepts. To decrease bias error, the GB Algorithm is often utilized. Both prediction and classification techniques may benefit from the gradient-boosting approach. MSE is the cost function in a regression problem, and function Loss is the cost function in a classifier. GB Machine integrates results from many decision trees to create a final prediction. Take into account that in a gradient-boosting machine, each learning rate is a decision tree.

k-nearest neighbor: K-Nearest Neighbors (KNN) is a machine learning method that uses case similarities to classify data points. It is a non-parameterized approach that relies on comparing the characteristics of data points to determine their classifications. The process involves selecting a value for K, calculating the Euclidean distance between the K nearest neighbors, and then assigning a category based on the majority of the neighbors. The KNN algorithm identifies the closest data points in terms of feature similarity, with K representing the number of data points used in the analysis. The distance measure, typically Euclidean distance, and the corresponding values play a crucial role in the KNN classifier. By considering the K nearest neighbors and their associated labels, KNN can classify new data points in the feature space.

Logistic regression: The most widely used ML algorithm is logistic regression. It is a statistical technique that is also known as the Logit model. Depending on the dataset of independent factors, it calculates the likelihood of an event occurring. In binary logistic regression, there is just one binary dependent variable, coded by an indicator variable, with two parameters labeled 0 and 1, and the independent variables may be either binary or continuous. Important assumptions to keep in mind are that the dependent variable has to be categorized and that the independent variable should not be multi-collinear. The dependent variable is confined between 0 and 1 since the outcome is a probability. A logit transformation is performed to the odds in logistic regression, which is the likelihood of success divided by the probability of failure.

4. RESULTS AND DISCUSSIONS

In this section, the paper presents the results obtained after prediction. Section 4.1 describes dataset taken in this paper for evaluation. Section 4.2 describes the implementation details. Section 4.3 describes the data visualization and finally in section 4.4 results are analyzed with a comparative state-of-art.

4.1. DATASET DESCRIPTION

This paper introduces a software fault dataset designed for comparing bug prediction models [21]. The dataset includes information necessary for estimating bugs based on source code measures, historic measurements, and access to data. The dataset covers several software systems, including Eclipse, PDE, Equinox, Lucene, and Mylyn. It provides historical information, biweekly versions of systems, source code metrics, and post-release defect numbers for each class. A sample of the dataset is presented in Table 3.

Table. 3. Dataset Representation

Dataset	Samples
Eclipse	5372
Equinox	325
PDE	1492
Lucene	692
Mylyn	1863

4.2. IMPLEMENTATION DETAILS

This paper has implemented and trained the models in the Keras framework with TensorFlow. The proposed model was trained using GPU on Google colab. Following the performance, parameters are used to evaluate the model's efficiency in terms of Accuracy, precision, and recall.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

$$\frac{Recall}{Sensitivity} = \frac{TP}{TP + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Where, TP = True Positive, FP= False Positive, FN= False Negative and TN = True Negative

4.3. EXPLORATORY DATA VISUALIZATION

Fig. 4 shows the feature correlation map using the eclipse dataset considering all the pre-determined conditions and variables and this map shows the correlation coefficient for different variables for all the possible pair of variables and helps in visualizing the given eclipse dataset.

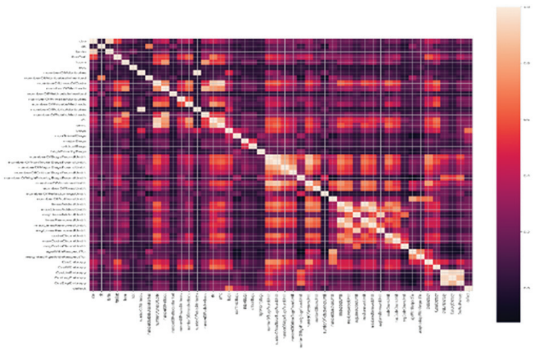


Fig. 4. Feature Correlation Map

4.4. RESULT ANALYSIS

Table 4 shows the accuracy, precision, and recall comparison of for eclipse dataset for different classifiers. The Accuracy of the Random Forest Classifier is maximum and it is minimum for the K-neighbors classifier. Precision is maximum for the Linear SVM classifier and minimum for the logistic regression classifier. The recall is maximum for the Decision Tree classifier and minimum for the Radial SVM classifier. Table 5 shows the accuracy, precision, and recall comparison of for equinox dataset for different classifiers. The Accuracy of Linear SVM, Random Forest, and Gradient Boosting is maximum and it is minimum for the K-Neighbors classifier. Precision is maximum for Linear SVM and Gradient Boosting classifier and minimum for logistic regression classifier. The recall is maximum for the Decision Tree classifier and minimum for the Decision Tree classifier. Table 6 shows the accuracy, precision, and recall comparison of for equinox dataset for different classifiers. The Accuracy of Linear SVM, Random Forest, and Gradient Boosting is maximum and it is minimum for the Logistic Regression classifier. Precision is maximum for different classifiers and minimum for Decision Tree classifiers. The recall is maximum for the Decision Tree classifier and minimum for Random Forest and K-Neighbors classifier. Table 7 shows the accuracy, precision, and recall comparison of for Mylyn dataset for different classifiers. The Linear SVM, Random Forest is maximum and it is minimum for Radial SVM classifier. Precision is maximum for the Linear SVM classifier and minimum for the K-Neighbors classifier. The recall is maximum for the Decision Tree classifier and minimum for the Gradient Boosting classifier. Table 8 shows the accuracy, precision, and recall comparison of for PDE dataset for different classifiers. The Accuracy of the Linear SVM Classifier is maximum and it is minimum for the Radial SVM classifier. Precision is maximum for the Linear SVM classifier and minimum for the Decision Tree classifier. The recall is maximum for 4 classifiers and minimum for the Logistic Regression classifier. Fig 5 represents the comparative state-of-art. In Fig 5, the paper compares with works presented by [16] and [23]. [16] proposed a Hyperparameter optimization algorithm and achieved an average of 87% accuracy whereas [23] proposed a dynamic parameter selection algorithm and achieved an accuracy of approx. 76%. From the result, we can observe that the proposed multi-constraint multi-optimization algorithm achieves better accuracy of approx. 99%.

Table 4. Parameter Comparison for Eclipse Dataset

Classifier	Accuracy	Precision	Recall
Decision Tree	99.39 %	66.66 %	97.47 %
Random Forest	99.47 %	95.85 %	96.46 %
Logistic Regression	86.07 %	46.4 %	96.46 %
Linear SVM	98.43 %	100 %	93.43 %
Gradient Boosting	99.18 %	56.93 %	26.26 %
K-Neighbors	84.66 %	97.94 %	22.72 %
Radial SVM	86.37 %	97.96 %	15.15 %

Table 5. Parameter Comparison for Equinox Dataset

Classifier	Accuracy	Precision	Recall
Linear SVM	97.53 %	100 %	100 %
K-Neighbors	80.24 %	100 %	93.54 %
Random Forest	97.53 %	86.95 %	93.54 %
Logistic Regression	80.23 %	74.14 %	93.54 %
Decision Tree	97.53 %	82.60 %	74.19 %
Gradient Boosting	97.53 %	100 %	64.51 %
Radial SVM	82.17 %	100 %	61.29 %

Table 6. Parameter Comparison Lucene Dataset

Classifier	Accuracy	Precision	Recall
Linear SVM	100 %	100 %	100 %
Logistic Regression	91.3 %	100 %	100 %
Radial SVM	90.17%	100 %	100 %
Decision Tree	100 %	83.3 %	100 %
Gradient Boosting	100 %	100 %	26.13 %
Random Forest	100 %	100 %	10.5 %
K-Neighbors	90.17%	100 %	10.5 %

Table 7. Parameter Comparison for Mylyn Dataset

Classifier	Accuracy	Precision	Recall
Linear SVM	99.78 %	100 %	98.8 %
Radial SVM	81.9 %	97.67 %	98.8 %
Decision Tree	99.57 %	87.87 %	98.8 %
K-Neighbors	83.4 %	63.33 %	98.8 %
Logistic Regression	87.12 %	98.8 %	34.11 %
Random Forest	99.78 %	66.66 %	22.3 %
Gradient Boosting	99.35 %	100 %	2.3 %

Table 8. Parameter Comparison for PDE Dataset

Classifier	Accuracy	Precision	Recall
Linear SVM	100 %	100 %	100 %
Random Forest	99.46 %	96.7 %	100 %
Gradient Boosting	100 %	55.17 %	100 %
Decision Tree	100 %	12.5 %	100 %
K-Neighbors	82.13 %	100 %	27.11 %
Radial SVM	82.66 %	31.8 %	11.86 %
Logistic Regression	85.066 %	100 %	1.6 %

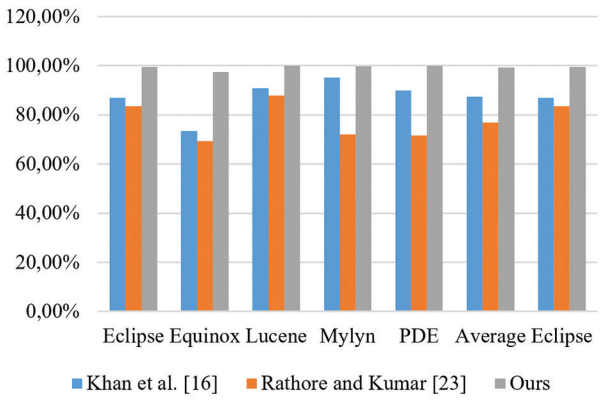


Fig 5. Comparative Accuracy Evaluation

5. DISCUSSION

Based on the information provided, the final result suggests that the proposed multi-constraint multi-optimization algorithm achieves a significantly higher accuracy of approximately 99% compared to other classifiers and state-of-the-art approaches. This is because the CCMOEO algorithm combines the power of evolutionary optimization with the incorporation of correlations between metrics and defect prediction. By leveraging these correlations, the algorithm aims to improve the accuracy of defect prediction in software and find better solutions to the constrained optimization problem. This indicates that the algorithm is highly effective in accurately classifying the datasets used in the study (Eclipse, Equinox, Mylyn, and PDE). The algorithm's superior performance in terms of accuracy, precision, and recall makes it a promising solution for the classification tasks considered in the study.

6. CONCLUSION

Software systems have a significant impact on society, and ensuring their trustworthiness is crucial. Bug-free software is a key factor in achieving trust, and reliability models are used to assess software reliability and predict faults. Researchers are exploring computational intelligence methods, including machine learning and optimization, to improve prediction models. In this paper, a software reliability prediction model is developed using a software metrics correlation multi-objective evolutionary algorithm-based model proposed for the identification of defect metrics for establishing the reliability of software. The result analysis was observed on seven classifiers and achieved an average accuracy of 99% which is approx. 13% improvement over comparative state-of-art. Future research can focus on training the algorithm on larger datasets, identifying failure points in AI software, and identifying weak reliable points that may lead to attacks.

Data Availability Statement:
All data are made available in the manuscript.

Conflict of Interest:
The authors declare no conflict of interest.

Funding Information: None.

7. REFERENCES

[1] A. K. Behera, M. Panda, S. C. Nayak, C. S. K. Dash, "An Artificial Electric Field Algorithm and Artificial Neural Network-Based Hybrid Model for Software Reliability Prediction", Computational Intelligence in Data Mining, Springer, 2022, pp. 271-279.

[2] X. Chen, Y. Shen, Z. Cui, X. Ju, "Applying Feature Selection to Software Defect Prediction Using Multi-objective Optimization", Proceedings of the IEEE

41st Annual Computer Software and Applications Conference, Turin, Italy, 4-8 July 2017, pp. 54-59.

[3] C. A. Coello Coello, S. González Brambila, J. Figueroa Gamboa, M. G. Castillo Tapia, R. Hernández Gómez, "Evolutionary multiobjective optimization: open research areas and some challenges lying ahead", Complex & Intelligent Systems, Vol. 6, No. 2, 2020, pp. 221-236.

[4] C. Diwaker et al. "A New Model for Predicting Component-Based Software Reliability Using Soft Computing", IEEE Access, Vol. 7, 2019, pp. 147191-147203.

[5] P. Dhavakumar, N. P. Gopalan, "An efficient parameter optimization of software reliability growth model by using chaotic grey wolf optimization algorithm", Journal of Ambient Intelligence and Humanized Computing, Vol. 12, No. 2, 2021, pp. 3177-3188.

[6] S. K. Dubey, B. Jasra, "Reliability assessment of component based software systems using fuzzy and ANFIS techniques", International Journal of Systems Assurance Engineering and Management, Vol. 8, No. 2, 2017, pp. 1319-1326.

[7] F. El Hajj Chehade, R. Younes, "Structural reliability software and calculation tools: a review", Innovative Infrastructure Solutions, Vol. 5, No. 1, 2020, p. 29.

[8] M. Esnaashari, A. H. Damia, "Automation of software test data generation using genetic algorithm and reinforcement learning", Expert Systems with Applications, Vol. 183, 2021, p. 115446.

[9] P. Gupta, M. K. Mehlawat, D. Mahajan, "Data envelopment analysis based multi-objective optimization model for evaluation and selection of software components under optimal redundancy", Annals of Operations Research, Vol. 312, No. 1, 2022, pp. 193-216.

[10] D. D. Hanagal, N. N. Bhalerao, "Literature Survey in Software Reliability Growth Models", Software Reliability Growth Models, Springer Singapore, 2021, pp. 13-26.

[11] S. Delphine Immaculate, M. Farida Begam, and M. Floramary, "Software Bug Prediction Using Supervised Machine Learning Algorithms", Proceedings of the International Conference on Data Science and Communication, 2019, pp. 1-7.

[12] G. Jabeen, P. Luo, W. Afzal, "An improved software reliability prediction model by using high precision error iterative analysis method", Software Testing, Verification and Reliability, Vol. 29, No. 6-7, 2019, p. e1710.

[13] A. Jaiswal, R. Malhotra, "Software reliability prediction using machine learning techniques", International Journal of Systems Assurance Engineering and Management, Vol. 9, No. 1, 2018, pp. 230-244.

[14] A. Jindal, A. Gupta, Rahul, "Comparative Analysis of Software Reliability Prediction Using Machine Learning and Deep Learning", Proceedings of the Second International Conference on Artificial Intelligence and Smart Energy, 2022, pp. 389-394.

[15] M. Job, S. Battista, R. Stanzani, A. Signori, M. Testa, "Quantitative Comparison of Human and Software Reliability in the Categorization of Sit-to-Stand Motion Pattern", IEEE Transactions on Neural Systems and Rehabilitation Engineering, Vol. 29, 2021, pp. 770-776.

[16] F. Khan, S. Kanwal, S. Alamri, B. Mumtaz, "Hyper-Parameter Optimization of Classifiers, Using an Artificial Immune Network and Its Application to Software Bug Prediction", IEEE Access, Vol. 8, 2020, pp. 20954-20964.

[17] T. M. Khoshgoftaar, Y. Liu, "A Multi-Objective Software Quality Classification Model Using Genetic Programming", IEEE Transactions on Reliability, Vol. 56, No. 2, 2007, pp. 237-245.

[18] P. Kumar, S. K. Singh, S. Deo Choudhary, "Reliability prediction analysis of aspect-oriented application using soft computing techniques", Materials Today: Proceedings, Vol. 45, 2021, pp. 2660-2665.

[19] Z. Li, M. Yu, D. Wang, H. Wei, "Using Hybrid Algorithm to Estimate and Predicate Based on Software Reliability Model", IEEE Access, Vol. 7, 2019, pp. 84268-84283.

[20] K. Lwin, R. Qu, G. Kendall, "A learning-guided multi-objective evolutionary algorithm for constrained portfolio optimization", Applied Soft Computing, Vol. 24, 2014, pp. 757-772.

[21] M. D'Ambros, M. Lanza, R. Robbes, "An extensive comparison of bug prediction approaches", Proceedings of the 7th IEEE Working Conference on

Mining Software Repositories, Cape Town, South Africa, 2-3 May 2010, pp. 31-41.

[22] P. Rani, G. S. Mahapatra, "Entropy based enhanced particle swarm optimization on multi-objective software reliability modelling for optimal testing resources allocation", Software Testing, Verification and Reliability, Vol. 31, No. 6, 2021, p. e1765.

[23] S. S. Rathore, S. Kumar, "Software fault prediction based on the dynamic selection of learning technique: findings from the eclipse project study", Applied Intelligence, Vol. 51, No. 12, 2021, pp. 8945-8960.

[24] S. K. Rath, M. Sahu, S. P. Das, S. K. Mohapatra, "Hybrid Software Reliability Prediction Model Using Feature Selection and Support Vector Classifier", Proceedings of the International Conference on Emerging Smart Computing and Informatics, Pune, India, 9-11 March 2022, pp. 1-4.

[25] K. Sahu, R. K. Srivastava, "Revisiting Software Reliability", Data Management, Analytics and Innovation, Springer, 2019, pp. 221-235.

[26] S. P. Sahu, B. R. Reddy, D. Mukherjee, D. M. Shyamla, B. S. Verma, "A hybrid approach to software fault prediction using genetic programming and ensemble learning methods", International Journal of Systems Assurance Engineering and Management, Vol. 13, 2022, pp. 1746-1760.

[27] Sangeeta, K. Sharma, M. Bala, "An ecological space based hybrid swarm-evolutionary algorithm for software reliability model parameter estimation", International Journal of Systems Assurance Engineering and Management, Vol. 11, No. 1, 2020, pp. 77-92.