Journal of Intelligent Systems and Internet of Things
Vol. 13, No. 02, PP. 150-165, 2024
DOI: https://doi.org/10.54216/JISIoT.130212

Software Testing Using Cuckoo Search Algorithm with Machine Learning Techniques

Deepashree N[1,*], M. Sahina Parveen[2]

[1]Research Scholar, Dept of Computer Science and Engineering, Dayananda Sagar University, Bengaluru, India
[2]Professor, Dayananda Sagar University, Bengaluru, India
Emails: deepashreeamcec@gmail.com ; Shahinaparveenm-cse@dsu.edu.in

Abstract

Software testing are any errors, flaws, bugs, mistakes, failures in a piece of software that might cause the programme to produce incorrect or unexpected results. Testing in software almost always increase both the time and money needed to finish a project. And finding bugs and fixing them is a laborious and expensive software process in and of itself. While it's unrealistic to expect to completely eradicate all testing from a project, their severity may be mitigated. It is possible to predict where bugs may appear in software using a method known as software defect prediction (SDP). The goal of each software development project should be to provide a bug-free product. Predicting where bugs may appear in code, often known as software defect prediction (SDP), is an important part of fixing software. Software of a high calibre should have few bugs. A software metric is a quantitative or qualitative evaluation of some aspect of the programme or its requirements. One of the more recent population-based algorithms, Cuckoo Search (CS) was inspired by the flight patterns of some cuckoo species as well as the Lévy flying patterns of other birds and fruit flies. The needs for international convergence are met by CS. KNN is a significant non-parameter supervised learning technique. This paper presents an overview of Stochastic Diffusion Search (SDS) in the form of a social metaphor to illustrate the processes by which SDS allots resources. The best-fit pattern identification and matching difficulties were addressed by SDS using a novel probabilistic method. As a multiagent population-based global search and optimization method, SDS is a distributed model of computing that makes use of interaction amongst basic agents. The behaviour of SDS is described by studying its resource allocation, convergence to global optimum, resilience, minimum convergence criterion, and linear time complexity within a rigorous mathematical framework, setting it apart from many nature-inspired search algorithms. This paper proposes a hybrid optimization strategy based on CS-SDS techniques. By using the global search strategy solution of the SDS algorithm, this hybridization idea aims to enhance the cuckoo bird's search strategy for the optimum host nest. To that end, the SDS method would be used to place the cuckoo egg in the most advantageous location. When compared to other classifiers, PC2's improved performance may be attributed to its higher recall values. When compared to the Naive Bayes and Radial Bias Neural Network classifiers, the KNN performs 7.64% and 2.20% better, respectively.

Received: October 21, 2023 Revised: February 23, 2024 Accepted: June 22, 2024

Keywords: Stochastic Diffusion Search; Cuckoo Search; Software Defect Prediction; K Nearest Neighbor; Naïve Bayes; Radial Bias Neural Network

1) Introduction

A problem in the software that causes it to malfunction in executable goods is referred to as a software fault. In the context of software engineering, a bug is an instance in which the programmed in question does not adhere to the criteria that have been specified for it. In common parlance, bugs are referred to as errors. The software engineers will be able to differentiate between the bugs in the software, the failures of the software, and the testing in the software. In the case of a failure, the software will not perform the actions that are anticipated by the users. The fault, on the other hand, is a hidden defect in the programming that might really materialize as a failure. This non-conformance to the requirements is generally referred to as the bug. There is a software quality model that

serves as a very helpful instrument in testing the many endeavours of different projects to satisfy the goals of the dependability of software. This model was developed. It is possible to utilize the metrics that are accessible early on in the data's lifespan in order to determine whether or not there is a need for additional monitoring of quality throughout the development process. Modelling in its many forms is used to determine which modules are most likely to have problems [1]Testing the programmed is going to be the key step that detects these bugs in the software throughout the process of developing software, so keep an eye out for it. In the event that a developer or a tester is able to correctly identify the flaws that exist in software, it is possible to cut down on the amount of money, time, and effort that are involved in the process of locating and fixing testing. The primary focus of this phase will be on the creation of a software system [2] of the highest possible quality.

Due to the fact that development is a labour-intensive activity, it is difficult to create software that is devoid of flaws; thus, it is essential to correct such flaws to the greatest extent feasible [3]. The Classification seeks for a new model set that either describes or differentiates the various data classes and ideas. Training samples for class labels that oversee the categorization of model learning will be provided by supervised learning, which is the process of organizing data into classes. The methods for classifying things make use of training sets, which are collections of things that already have their class labels attached to them. A model that has been generated once the classification algorithm has learned from the training set is called a "learned model." The new models will be classed using this model. The identification of fraud and the applications of credit risk will fit this sort of investigation. There are several different classification strategies that have been proposed for the purpose of the prediction of software detection [4].

The Software Defect Prediction (SDP), an educational challenge that also has an academic relevance. A static code will attribute from the log-based software, and it will then release a model that was constructed for forecasting the bad modules that will be included in its subsequent release. This will detect the faulty areas of the programmed, which is beneficial when finances are tight, and the system is extremely vast for testing on a thorough level. The predictor of testing will make certain that the software developers [5] examine the areas of the programmed that are most likely to have errors. They further assure the quality of software, reduce the costs of delivery, and incorporate defect prediction data sets from real-world projects. Additionally, they assist academics in the construction of similar models.

2) Related Work

The work approximations of a project and the fault estimations of a software segment have both been improved as a result of [6]. These studies commonly shown conclusion diversity when taking into consideration the productive actions for a variety of projects or modules. Because of this, there is a question about the generalizability of the conclusions drawn from the empirical software effort (SE). Researchers in SE are required to conduct experiments to determine whether or not their conventional findings were valid for the data subsets they were analysing. As a result, empirical SE has been useful in the search for nearby locations with features that were similar.

[7] have performed an analysis of variance on the two primary stratification alternatives for SDP, which are under-sampling and over-sampling (ANOVA). By using a factorial strategy, the research safeguards a significant number of recently collected SDP datasets. At a significance level of 0.05, the author found that both the principal under-sampling effect and the interaction between under- and over-sampling had a significant impact on the results. However, the most significant results of oversampling turned out to be irrelevant.

[8] have developed methods for dataset class boundary preservation using the privatisation technique. In the case of CLIFF, pruner was used to get rid of unconnected occurrences. The MORPH programme was a data mutator that moved the data around across a random distance while keeping in mind that it should not violate any class boundaries. In a CCDP work experiment, CLIFF+MORPH compared and contrasted ten different defective datasets taken from the 28 PROMISE data pool. Based on this study, we know: 1) based on the effectiveness calculated by fault prediction, it was shown that CLIFF+MORPH outperformed well substantially. 2) The CLIFFed+ MORPH algorithms give better privacy than the state-of-the-art privacy algorithms. Wang and Yao (2013) have analysed numerous different types of class Imbalance Learning methods, including threshold shifting, ensemble algorithms, and resampling strategies. AdaBoost was one of the methods that were investigated in this study. The Negative Correlation (NC) measure was shown to have the best performance based on metrics like the Mean, the Area under the Curve, and the Balance (AUC). Because it did not need the pre-definition of any additional parameters, it was able to prove to be just as efficient and successful as the real AdaBoost. NC. SDPs have been presented by [9]. These SDPs help to improve the software confirmation and confirmation activities by providing crucial data to experiment allocating resources and releasing of planning choices. This author had conducted a survey of the software development and experimenting procedure used by two large companies operating in the automotive and telecom industries. From this, they were able to map several testing predicting techniques and their application in the respective lifecycle phases of these companies' products. Depending on the

survey and the already established trends, likely directions were also detected for SDP approaches and their use in such sectors. Dictionary learning has been shown to be an effective strategy for predicting software [10] by Jing et al. (2014). As a result, we have developed a method called Cost-sensitive Discriminative Dictionary Learning (CDDL) for the classification and estimate of software errors. The significantly used data sets from NASA programmes were utilised as experimental data in order to calculate the performance of each of the compared techniques. The results of the experiments demonstrated that CDDL executed a large number of approaches that are indicative of the most recent advances in fault prediction [11].

[12] conducted a study to investigate the effects of applying a fuzzy multilevel technique to the task of ranking software dependability indicators. When evaluating the performance of safety-critical digital systems, reliability was an important factor to consider. The features of the digital safety-critical systems were either expressly or indirectly reflected by the precautions taken by software engineering. Therefore, using these measurements, patterns might be created to forecast the predictability of software programmes that operate on safety-critical systems. Because it was not necessary for every software engineering metric to have contributed to the prediction of the dependability, they needed to rank based on the reliability. These expert evaluations were compiled, and then Chen's fuzzy logic based ranking approach was used to rank them. The use of fuzzy set theory [13] was found to be particularly suited for these conditions due to the fact that the data associated with this problem were inherently inaccurate. After further development, a model to forecast the dependability of safety-critical digital systems was improved with the help of the software engineering measures that placed highest in the rankings.

The malfunction of software [14] inside an executable product is referred to as a software fault. The failure of a piece of software to meet its specified criteria is known as a bug in software engineering. Software testing, failures, and bugs are three terms that software engineers should be able to distinguish between one another. A software bug is a flaw or a hidden programming issue that may or may not reveal itself as a failure. Software failure occurs when the software loses control and does not do the actions that the user anticipates it would. Creating a high-quality software system is an endeavour that is not only tough but also incredibly costly. Optimization [15] is the most useful field in mathematics and computer science since the majority of issues that people face in real life can be categorised as some kind of optimization issue. The mathematical connection that exists between an issue's objective function, possible restrictions, and decision variable might provide insight into how difficult the task is. Combinatorial issues (also known as discrete problems) and continuous problems (also known as global optimization) are both examples of difficult optimization problems. Continuous optimization [16] problems may be limited or uncontrolled (bound constrained). In the field of machine learning, feature selection is often used when the learning job at hand comprises attribute datasets with a high dimension and a high level of noise. The vast majority of the feature selection algorithms make use of local search throughout the whole of the process. As a consequence of this, it is challenging to arrive at solutions that are near ideal or optimal. In metaheuristic optimization, a solution may be located in the complete search space, and a global search ability is employed. This considerably boosts its capacity to find high-quality solutions in a given amount of time, and it also makes it possible to find solutions in the full search space. The SDP issue that arises in medical software is the focus of this paper.

3) Proposed Framework

The research that had been done up to this point had focused on the metrics that define the code modules as well as the learning techniques that are used to develop fault prediction models. Data mining methods, in conjunction with machine learning algorithms, have been used in the process of defect prediction in software and are also utilized in software repositories. Knowledge Discovery in Databases, abbreviated as KDD, is a multi-step process that includes data selection, pre-processing, transformation, mining, interpretation, and assessment. The data mining will serve as a fundamental activity in the KDD [17], and these procedures will be carried out with the assistance of specialists in order to construct the SDP models. A The process of extracting information from a huge amount of data is known as data mining. This process involves activities such as categorization, clustering, regression, and association. The method of categorization, which involves organizing data into distinct categories, is the primary topic of discussion in this article [18].

Bayes' Theorem:

$$P(R \mid S) = \frac{P(S|R) \cdot P(R)}{P(S)} \quad (1)$$

Where:

- $P(R \mid S)$ is the posterior probability of hypothesis R given evidence S.
- $P(S \mid R)$ is the likelihood of evidence S given hypothesis R.

- $P(R)$ is the prior probability of hypothesis R.
- $P(S)$ is the prior probability of evidence S.

In the models of defect prediction, the techniques of machine learning have been used for the purposes of learning and predicting the defective modules that are present in the programmed. The machine learning algorithms will be comprised of the software metric data together with the data formats [19]. That system is referred to as a learning system, and it is described as the system that learns from its experience in relation to a class of tasks and their performance measure, which is the performance at its task. Additionally, the learning system strives to enhance its experience. The data set in this is split into two parts: the training data set and the testing data set. These are done so that a learning system may be designed [20]. Support Vector Machines (SVM) Margin Equation:

$$\min_{\mathbf{w},b} \frac{1}{2} \| \mathbf{w} \|^2 \tag{2}$$

Subject to:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \; \forall i \tag{3}$$

Decision Tree Splitting Criteria (Gini Index):

$$\text{Gini}(t) = 1 - \sum_{i=1}^{c} p_i^2 \tag{4}$$

Where p_i is the probability of class i in node t.

Certain predictive functions will be created, and then they will be trained on the training data set. The testing data set will then be used to assess the outcomes of the training process. The decision trees, the neural network, often known as NN, and the Bayesian belief network are all examples of machine learning methods that are used in the process of predicting software testing (BBN). For the construction of the prediction models, a number of different machine learning strategies, such as the supervised, the semi-supervised, and the unsupervised, are used. The supervised learning strategy is by far the one that sees the greatest use overall. Artificial Neural Network (ANN) Activation Function (e.g., Sigmoid):

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{5}$$

K-Nearest Neighbors (KNN) Classification:

$$\hat{y}(\mathbf{x}) = \text{mode}(y_{\text{nearest}}) \tag{6}$$

Where y_{nearest} are the labels of the k nearest neighbors of $\mathbf{x}$.

Ensemble of decision trees where each tree is trained on a bootstrap sample of the data and a random subset of features.

Cross-validation Accuracy:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{7}$$

The Machine Learning Classification Algorithm is that popular approach in the SDP which categorizes the code attributes of software which is made into defective or non-defective that is completed using a classification model from the data of software metrics in the development projects. This is done using a classification model from the data of software metrics in the development projects. In the event that an error is detected during system testing from field tests, the fault data of the module is either marked with a 1 or left unmarked if the issue was not reported. The variable predictors of this model will be constructed with the help of the failure data and software metrics that were obtained before. Logistic Regression Hypothesis:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \tag{8}$$

Naive Bayes Classifier:

$$P(C_k \mid x_1, \ldots, x_n) = \frac{P(C_k) \prod_{i=1}^{n} P(x_i|C_k)}{P(x_1,\ldots,x_n)} \tag{9}$$

Regression Analysis (Linear Regression):

$$y = \beta_0 + \beta_1 x + \epsilon \tag{10}$$

Clustering (K-Means Algorithm):

$$\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mu_i\|^2 \quad (11)$$

Where μ_i is the centroid of cluster i. The SDP makes use of a number of different classification methods, some of which include decision trees, logistic regression, naive Bayes, and NNs. These techniques are all mentioned in the previous sentence. But the SDP is a highly unsolvable problem, and comparisons with the result of benchmarking the outcome of the defect prediction by applying machine learning classifiers show that there are no major differences found that will perform across all of the datasets. In large-scale software systems, there is a need for a model that can forecast the occurrence of testing.

NNs refers to a network of linked neurons in which the information transmission takes place as a result of the firing of electrical pulses across the connections between the neurons. The number of incoming pulses that are necessary for the activation of a neuron is reduced or increased depending on the circumstances. Learning will be possible for the NNs because to their behavioral characteristics. Supervised learning, unsupervised learning, and reinforced learning are the three fundamental learning algorithms that we uncover. Learning with a teacher is another name for supervised learning, and it refers to the process by which a goal value is assigned to each of the inputs that a person is training with. Supervised learning is a technique that is often used.

A replication of a biological brain system, an Artificial Neural Network (ANN) is a paradigm for the processing of information that is modelled after neural networks seen in living organisms. An NN that has received training may be considered an authority on the information category, and this is provided for the purpose of analysis. In the ANN, there have been a total of three levels, the first of which is the Input Layer. This layer is comprised of input units, each of which represents raw information that is supplied for this network. The Hidden Layer is represented by employing the hidden units, which have been impacted by the behavior of the input units, and the weights will link the input units with the hidden units. This layer is a representation of the hidden layer. The behavior of the output unit will be contingent on the particularity of these hidden units as well as the weights that link the hidden output units with the output units.

Precision and Recall:

$$\begin{aligned} \text{Precision} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \\ \text{Recall} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \end{aligned} \quad (12)$$

Gradient Descent:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad (13)$$

F1 Score (Harmonic Mean of Precision and Recall):

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (14)$$

Principal Component Analysis (PCA):

$$\mathbf{z} = \mathbf{W}^T \mathbf{x} \quad \textbf{(15)}$$

Where **W** are the principal components? The fundamental strategy here is to convert a computation issue using functions of categorization into a grade two programming problem with restrictions. The kernel function is employed in this situation because it can register an input data set to that of a higher multi-dimensional feature space. This allows the sample to be separable in the feature space, which would not have been possible without the kernel function. This particular SVM has the following essential characteristics: 1) It will operate in those spaces having large dimensions and will be under little learning samples; this just indicates that the capability of the support vector machine's to learn is independent of the feature of the space dimension. 2), which is a grade two formula and is a desired global solution. In other words, it is a grade two formula. 3) that connects relevant non-linear models with several additional methods

A decision tree is a kind of decision support tool that may take the form of either a tree-like model or a graph of choices and the consequences of those decisions. These repercussions can include the outcomes of random events, the costs of resources, and utility. This is one way that an algorithm may be shown, and it is useful for determining which strategies have the potential to bring about the desired outcome. In actuality, they can only be chosen based on the probability model since it is the best option model available via the online selection process. The decision trees will also be used as a descriptive tool for the computation of conditional probabilities, which is one additional application for them.

The decision trees that are a part of the decision assistance tools have several benefits to offer:

1. It is vital to recognize that even with some concrete facts, the most relevant insights are derived from the description of circumstances, their associated costs and options, as well as the probabilities and preferences towards the outcomes.
2. Employing a "white box" model that can be reproduced with elementary mathematical operations.
3. Is integrated with the use of many additional decision-making methods
4. The Bayes theorem, which was devised by the Reverend Thomas Bayes, a mathematician who lived in the eighteenth century, serves as the foundation for the Bayesian belief network.

In this equation, P (R/S) represents a posterior probability of the hypothesis, P (S/R) represents the likelihood of the data, and P (R) represents the prior probability of the hypothesis. Equation (3.1) illustrates the fundamental Bayes rule, which may be viewed in terms of the process of revising one's belief on the hypothesis R in light of fresh data. S. the posterior belief P (R) by its probability P (S/R) that the event S will occur if the hypothesis R is correct.

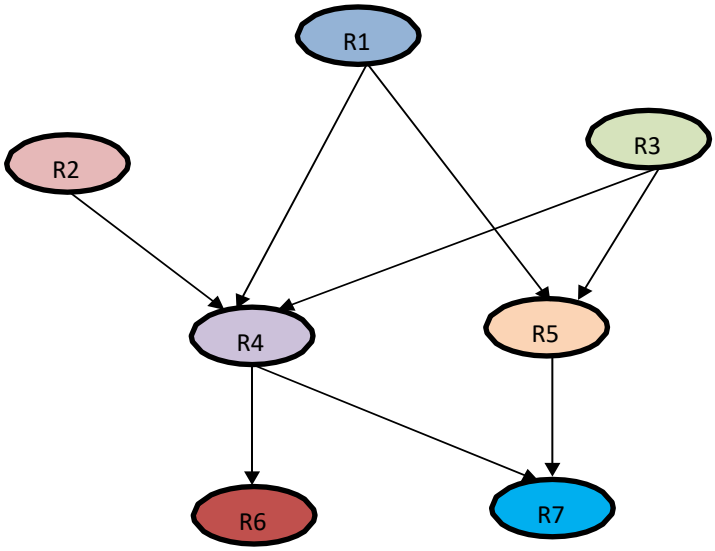

Figure 1. Example of BBN

From the figure 1 , BBN network indicated the packets transmission from one node to another node, R1 to R7 represents the nodes in the system. The BBN is a Directed Acyclic Graph (DAG), as illustrated in figure.1, in which the arcs indicate dependencies, which are the cause-effect interactions in the variables, and the nodes represent system variables. There is a correlation between the condition of the nodes and the likelihood. It has been determined what the likelihood is for a root node, and this probability is an a priori value that is estimated by inference to other nodes. The BBN may be thought of as a method for approaching graph theory and probability theory. Radial Basis Function (RBF):

$$\phi(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}-\mu\|^2}{2\sigma^2}\right) \tag{16}$$

Entropy (Information Gain in Decision Trees):

$$H(T) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{17}$$

Chernoff Bound (Probability Theory):

$$P(X \geq t) \leq \inf_{s>0}\left(\frac{E[e^{sX}]}{e^{st}}\right) \tag{18}$$

The Random Forest, often known as RF, is an effective novel method for data exploration, data analysis, and predictive modelling. This accomplishes the task of error identification in addition to the construction of robust

models of prediction. Random selection is going to be done on a small portion of the characteristics using the RF method. Because of this, the node that contains the feature that has the best variable will be divided. Utilizing a feature subset that is based on correlation will allow this embedded classifier, which is the enhanced RF, to achieve a higher level of precision. In order to get a higher level of accuracy in the assessment of performance parameters such as accuracy, specificity, and sensitivity, a 10-fold cross-validation of the data is carried out.

4) METHODOLOGY

There are many different SDP strategies that may be used, some of which include machine learning, parametric, mixed model, and machine learning methodologies. The most recent studies have shown that the researchers employ machine learning in order to forecast the quality of the programmed. These days, the classification and clustering techniques to machine learning are the ones that see the most widespread use. It is necessary for the data or characteristics to play a significant role in order to have an efficient defect prediction model. The K Nearest Neighbor (KNN) is a non-parameter of a supervised learning algorithm that is considered to be particularly essential. Without using any extra information, the rules of classification may be derived from the training samples alone.

Algorithm 1: Overall Working Model of Proposed work

1. Collect software metrics data
2. Preprocess data:
   - Handle missing values
   - Normalize or standardize features
   - Encode categorical variables if necessary
   - Split data into training and testing sets

3. Select relevant features
4. Engineer new features if needed

5. Choose machine learning algorithms
6. Train models on the training data
   - Initialize model parameters and hyper parameters
   - Optimize using techniques like grid search or randomized search

7. Evaluate model performance
   - Calculate metrics: Accuracy, Precision, Recall, F1-score, ROC-AUC
   - Perform cross-validation

8. Predict defects in software modules
   - Apply trained models to testing data

9. Enhance prediction models

The KNN classification method will make a prediction about the test sample and its category by using the K training samples that are the closest neighbors to those samples. This prediction will be based on the K training samples. Classifiers such as Naive Bayes, KNN, and the Radial Basis Function are used in this system in order to categories the numerous flaws that have been identified (RBF). During the course of the algorithm assessment, the PC1 and PC2 Datasets were used on a regular basis. The flowchart depicting this suggested technique may be seen in figure 2.

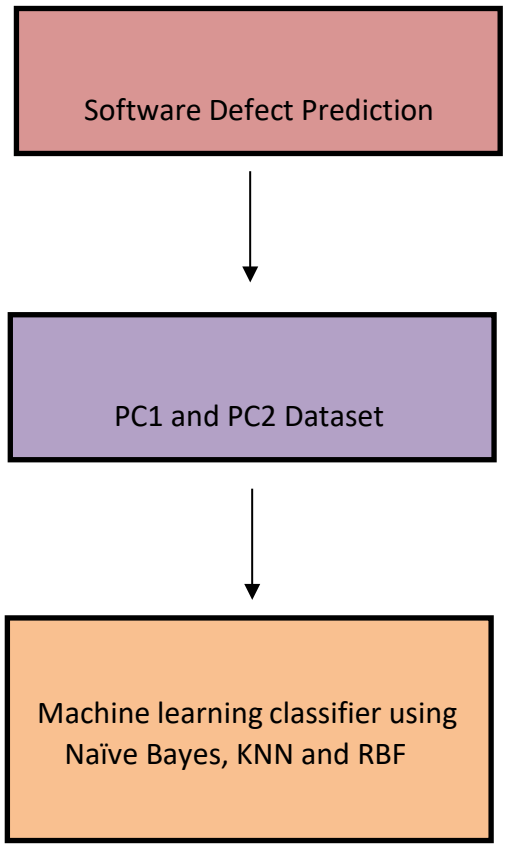

Figure 2. Flowchart of Proposed Methodology

This paper advocates using the CS-SDS approach as a means of addressing the drawbacks associated with the CS. The mission of the CS is to come up with fresh ideas that might be more successful than the outdated strategies that are now being used to deal with the existing nest population. The evaluation of the quality of the maximum solutions is done with the help of the objective function of the issue that has to be solved. Nevertheless, there are also certain issues that arise in the actual world in which the least possible values of a particular objective function are the focus of attention. The transformation of these sorts of issues from problems involving minimization to problems involving maximisation using the equation min (f (x)) = max (–f (x)) is a piece of cake from a mathematical point of view. Instead of being called an objective function, this sort of altered function is now called a fitness function.

CS makes an effort to identify desired solutions that are also successful for a number of situations involving continuous optimization. On the other hand, it might be difficult at times, particularly when the appropriate answers cannot be established for other optimization issues that are similar to the proverb "there is no such thing as a free lunch." By adding hybridization to optimization algorithms, which help solve a specific set of issues, it is possible to sidestep this issue and find a solution to it. As a result, computer science has been combined with a variety of different machine learning approaches, optimization issues, and so on. Hybridization allows for the application of each and every component of CS. Initialization process, evaluation function, movement function, and the others are some examples of those that are tried here. The SDS will investigate and look for the model that provides the greatest possible match in a certain search area. For example, one particular word included inside a particular text document would serve as the model, while the content itself would serve as the search space. During the search process, each agent will function alone, and the only time they will communicate with one another is when they will share information about what has been found. The SDS does not take into account the amount of time needed for an agent to get to its hypothesis point; instead, it focuses only on spatial concerns such as the magnitude of the search space. Having said that, after the selection of the hypothesis, there will be a cost associated with assessing the same, and this cost is known as the cost of the test function. During the SDS process, each agent has the ability to access the whole search area and is tasked with conveying information on the target prototype's level of completion.

- **PC1 and PC2 Dataset**

The PC1 project is Earth-orbiting satellite software system the values of which are shown in Table 1.

Table 1: Modules and requirements in PC1 dataset

	Total modules	Modules with testing	Modules have req.	Defect modules with req.	Total # of req.	Req. related to module(s)	# Of req. related to testing
PC1	1107	73	203	44	320	320	109

The PC2 data collection has a total of 5589 data points, each of which includes module product parameters as well as the defect data connected with it. Only 23 of the data points, or 0.4 percent of the total number of data points, had the 'defective' label attached to them. Table 2 displays the dataset's descriptive statistics, which can be found for PC1.

Table 2: Descriptive Statistics for PC1 Dataset

	Minimum	Maximum	Mean	Std. Deviation	Skewness	Kurtosis
					Statistic	Statistic
loc	0.0	602.0	23.376	35.2840	7.567	99.119
vg	1.0	136.0	5.511	8.9590	6.415	64.798
evg	1.0	123.0	2.767	5.5677	11.957	219.777
ivG	1.0	123.0	3.321	6.4020	10.260	149.776
N	1.0	2785.0	117.393	197.3369	6.056	58.178
V	0.00	25942.69	699.7112	1509.54568	8.454	109.684
L	0.00	2.00	.1294	.14695	3.884	30.113
D	0.00	270.66	15.3963	16.33751	5.204	59.534
I	0.00	598.33	32.9045	35.38768	5.795	67.149
E	0.00	4279633.01	28822.8824	170643.60333	18.550	410.488
B	0.00	8.65	.2352	.50477	8.379	108.253
T	0.00	237757.40	1601.2730	9480.19999	18.550	410.488
lOCode	0	600	22.43	33.575	7.793	106.530
lOComment	0	159	4.70	10.518	5.365	50.839
locCodeAndComment	0	48	.94	3.345	8.324	90.717
lOBlank	0	225	6.75	12.301	7.428	101.622
uniq_Op	1.0	99.0	13.308	8.1822	2.172	13.730
uniq_Opnd	0.0	538.0	20.893	29.0514	7.997	116.630
total_Op	1.0	1641.0	66.493	111.7032	6.548	68.712
total_Opnd	0.0	1144.0	50.902	86.3087	5.507	46.414
branchCount	1.0	236.0	9.577	16.5407	5.972	54.399
Valid N (listwise)						

Frequency vs Lines of Code (LOC), VG, EVG, IVG, N, V, L, D, I, E, B, T, IO Code, IO Comment, IO Code and IO Comment, IO Blank, uniq- Op, uniq-Opnd, total-Op, total-Opnd, branch count and testing for PC1 dataset as shown in Figure 3 to 6.

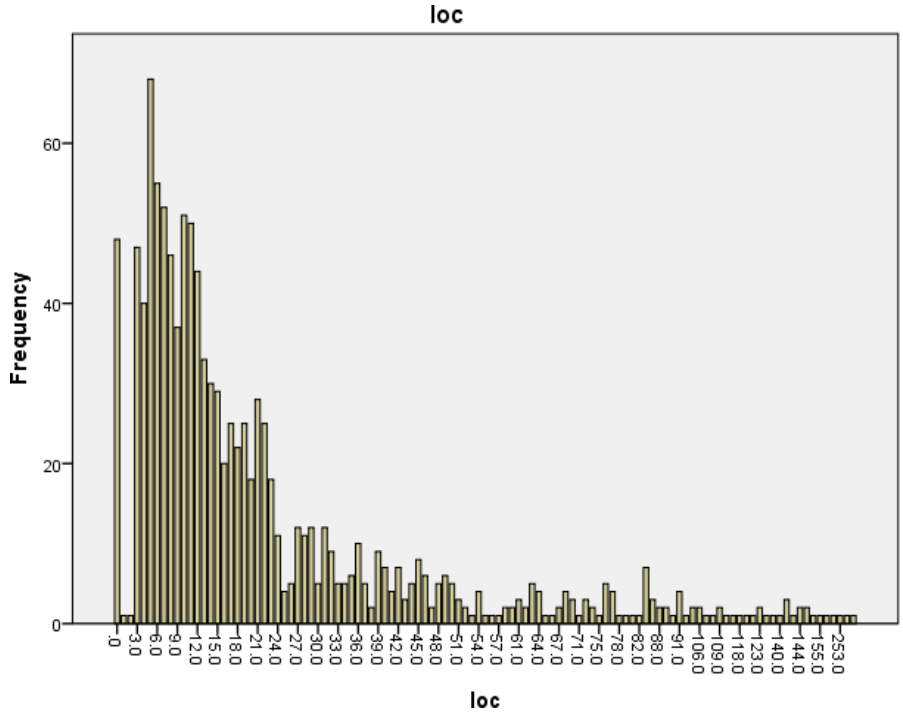

Figure 3. Lines of Code (LOC) for PC1 Dataset

Figure 3 shows that when Lines of Code (LOC) increases, the frequency gets decreased.

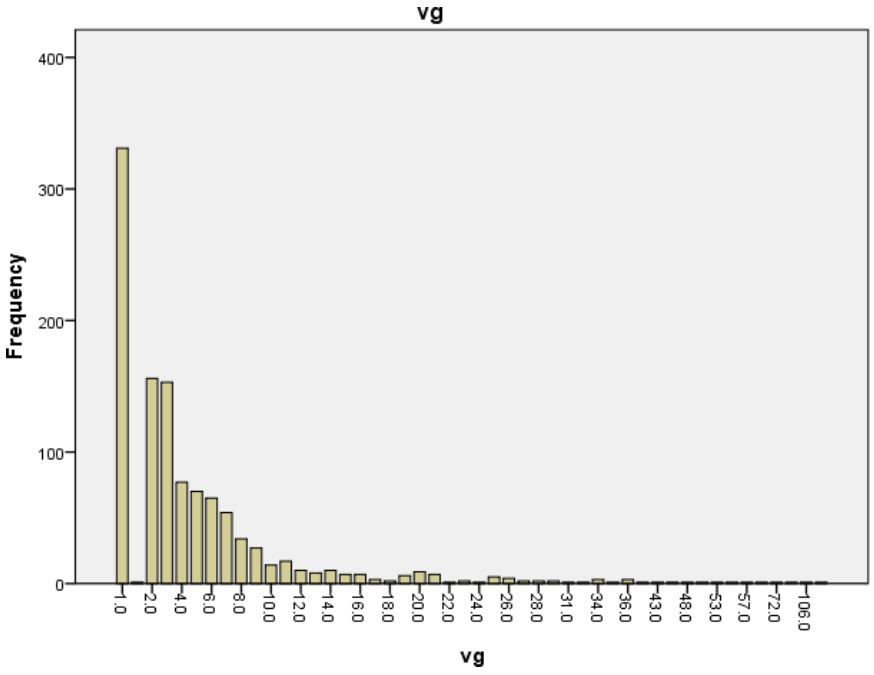

Figure 4. Cyclomatic Complexity (VG) for PC1 Dataset

Figure 4 shows that when Cyclomatic Complexity (VG) increases, the frequency gets decreased.

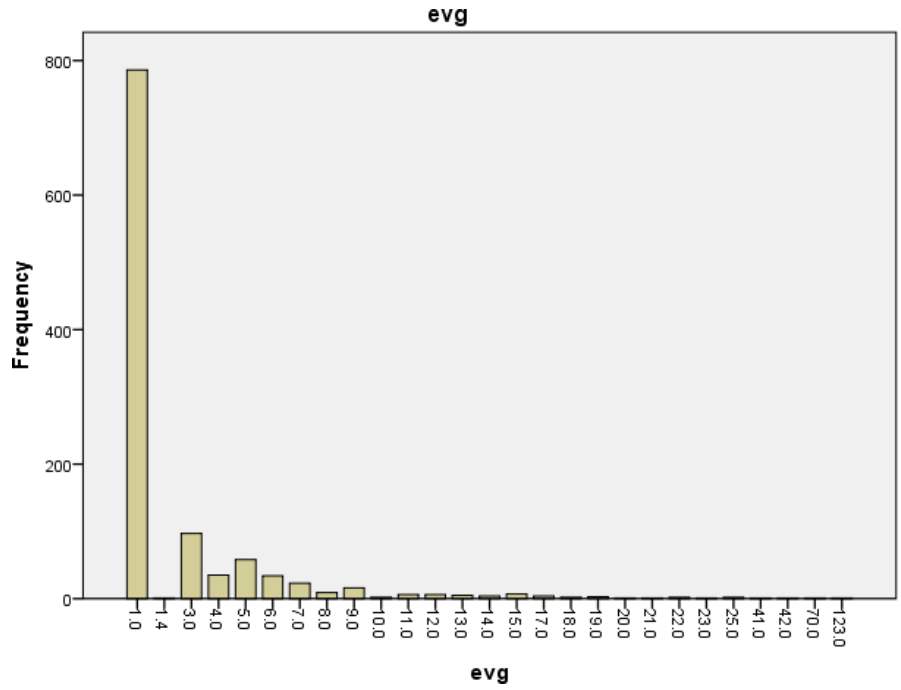

Figure 5. Extended Cyclomatic Complexity (EVG) for PC1 Dataset

Figure 5 shows that when Extended Cyclomatic Complexity (EVG) increases, the frequency gets decreased.

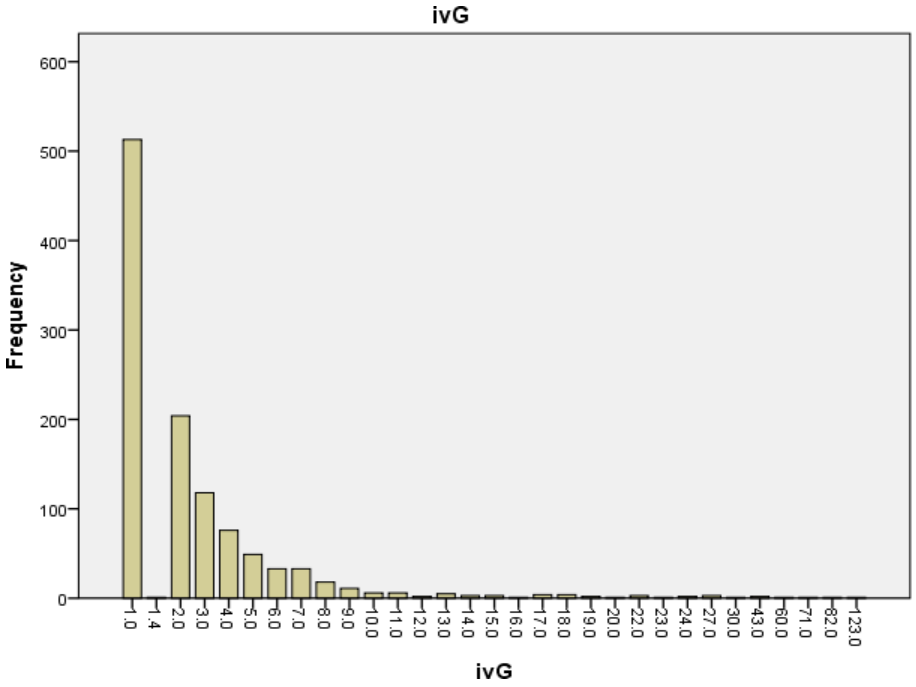

Figure 6. Design Complexity (IVG) for PC1 Dataset

Figure 6 shows that when Design Complexity (IVG) increases, the frequency gets decreased.

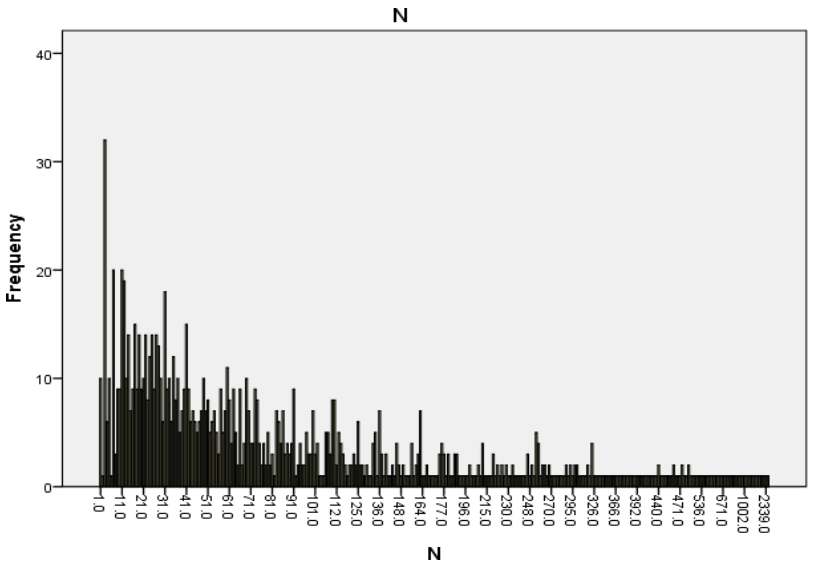

Figure 7. Total Operators and Operands (N) for PC1 Dataset

Figure.7 shows that when the Total Operators and Operands (N) increases, the frequency gets decreased

5) Results and Discussion

In this work, PC1 and PC2 Datasets are used with classifiers such as Naïve Bayes, KNN and RBF. Experiments are conducted by using PC1 Dataset and PC2 Dataset. Figure 8 to 9 shows the resulting graph and table 3 to 6 show the resulting values for classification accuracy, Root Mean Square Error (RMSE), precision and recall.

Table 3: Classification Accuracy

	PC1 dataset	PC2 dataset
Naïve Bayes	89.17	90.73
kNN	92.9	95.83
RBF	92.69	97.85

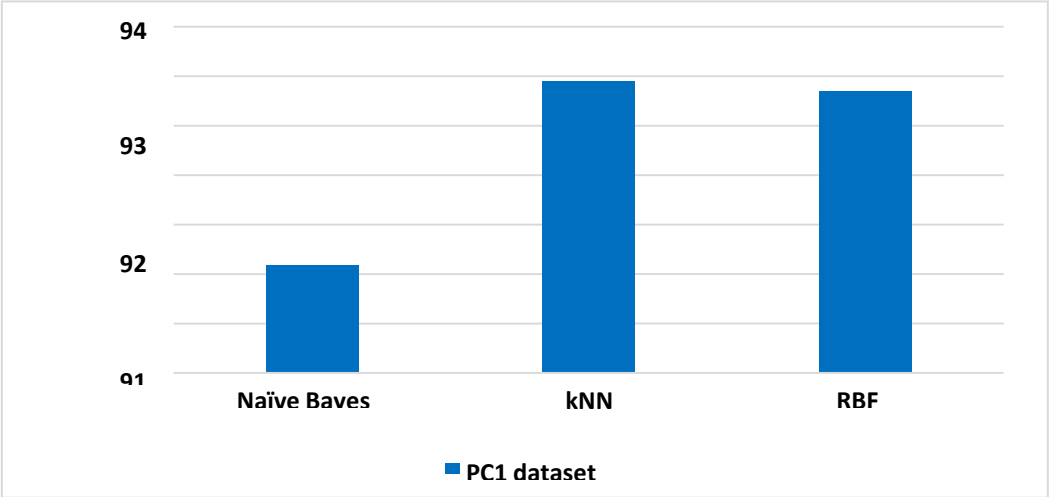

Figure 8. Classification Accuracy for PC1 Dataset

Figure 8, shows that the classification accuracy of KNN in PC1 dataset achieved better performance when compared to various classifiers. The RBF is better by 4.1% and by 0.23% than Naïve Bayes and RBF classifiers respectively.

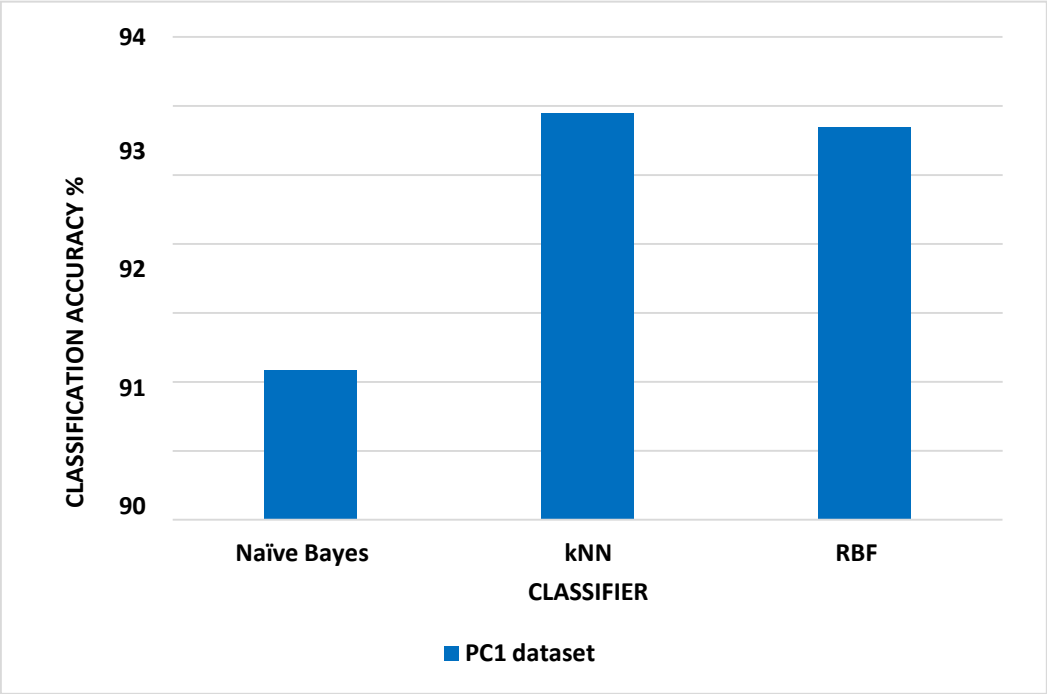

Figure 9. Classification Accuracy for PC2 Dataset

Figure 9, shows that the classification accuracy of RBF in PC2 dataset achieved better performance when compared to various classifiers. The RBF is better by 7.6% and by 2.1% than Naïve Bayes and KNN classifiers respectively.

Table 4: Root Mean Square Error (RMSE)

	PC1 dataset	PC2 dataset
Naïve Bayes	0.3255	0.2981
kNN	0.2826	0.2037
RBF	0.2462	0.145

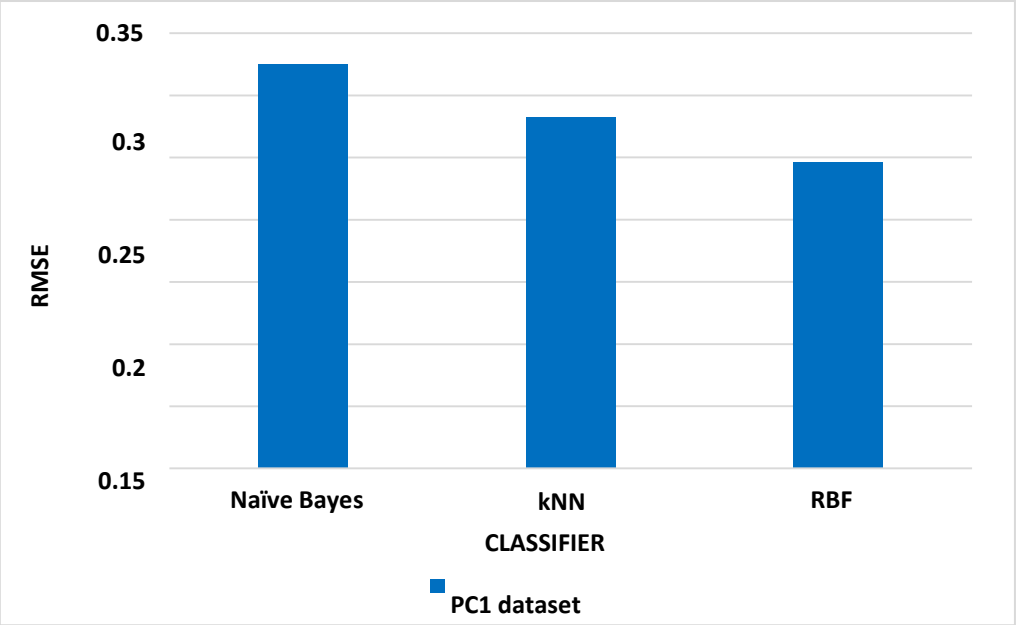

Figure 10. Root Mean Square Error (RMSE) for PC1 Dataset

Figure 10, shows that the RMSE of RBF in PC1 dataset achieved better performance by lowering Root Mean Square Error values when compared to various classifiers. The RBF is better by 27.7% and by 13.77% than Naïve Bayes and KNN classifiers respectively.

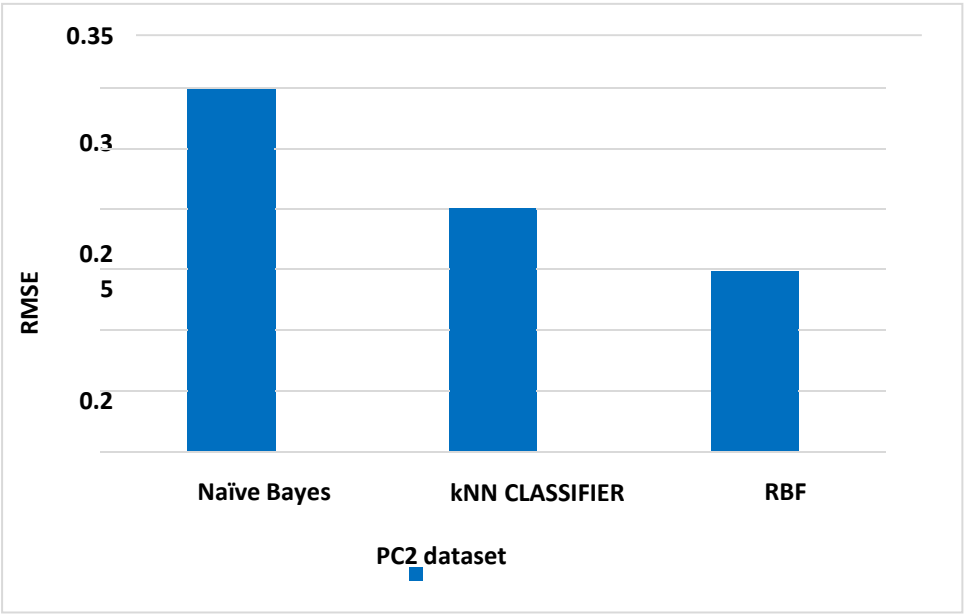

Figure 11. Root Mean Square Error (RMSE) for PC2 Dataset

Figure 11, shows that the RMSE of RBF in PC2 dataset achieved better performance by lowering Root Mean Square Error values when compared to various classifiers. The RBF is better by 69.1% and by 33.7% than Naïve Bayes and KNN classifiers respectively.

Table 5: Precision

	PC1 dataset	PC2 dataset
Naïve Bayes	0.899	0.959
KNN	0.922	0.957
RBF	0.878	0.958

The performance of machine learning systems for SDP can be considerably improvised by making use of classifiers like naive Bayes, KNN, and RBF. PC1 and PC2 datasets have been used for conducting experiments. It has been shown that RMSE is lowered by using PC2; PC2 also performs better for recall and classification precision. In terms of precision, compared to the PC1 dataset, naive Bayes for PC2 performs better by 1.73%. Compared to PC1, KNN for PC2 is better by 3.105%. RBF is better by 5.41% for PC2 dataset. Overall performance improved by RBF by 7.55% than Naïve Bayes and by 2.085% than KNN

6. Conclusion and Future Scope

The cornerstone of this study effort is the indisputable need for effective risk management in the software development process in order to prevent the failure of individual projects. An in-depth analysis of risk management and the software development life cycle from the years 2001 to 2016 was carried out with the purpose of determining whether or not risk-based testing is necessary. This research has investigated a number of tools, models, and techniques that are used for risk management, risk-based testing, and software testing. These investigations supported the identification of an intelligent vendor that provides risk-managed testing in a number of different evolutionary environments. Due to the fact that the research gaps revealed a demand for risk management and testing in evolutionary computing environments, this study investigated a variety of methods, models, and tools for risk management in artificial intelligent environments, pervasive environments, big data environments, cloud environments, and internet of things environments. Test case prioritisation, which in turn helps increase fault detection rates, is facilitated by HDBNN's identification of risk classes. In both risk-based testing methodologies, the study demonstrated a link between risk, the software testing life cycle, and quality assurance for evolutionary software project. All of the goals of the study have been accomplished, and all bases have been adequately covered.

References:

[1] Alzahrani, A. O., & Alenazi, M. J. (2023). ML-IDSDN: Machine learning based intrusion detection system for software-defined network. Concurrency and Computation: Practice and Experience, 35(1), e7438.

[2] Praveen, P., Nischitha, M., Supriya, C., Yogitha, M., & Suryanandh, A. (2023). To Detect Plant Disease Identification on Leaf Using Machine Learning Algorithms. In Intelligent System Design (pp. 239-249). Springer, Singapore.

[3] Panigrahi, R., Kuanar, S. K., & Kumar, L. (2023). Method Level Refactoring Prediction by Weighted-SVM Machine Learning Classifier. In Mobile Application Development: Practice and Experience (pp. 93-104). Springer, Singapore.

[4] Durelli, V. H., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R., & Guimaraes, M. P. (2019). Machine learning applied to software testing: A systematic mapping study. IEEE Transactions on Reliability, 68(3), 1189-1212.

[5] Murphy, C., Kaiser, G. E., & Arias, M. (2007). An approach to software testing of machine learning applications.

[6] Briand, L. C. (2008, August). Novel applications of machine learning in software testing. In 2008 The Eighth International Conference on Quality Software (pp. 3-10). IEEE.

[7] Baskiotis, N., Sebag, M., Gaudel, M. C., & Gouraud, S. D. (2007, January). A Machine Learning Approach for Statistical Software Testing. In IJCAI (pp. 2274-2279).

[8] Noorian, M., Bagheri, E., & Du, W. (2011, July). Machine Learning-based Software Testing: Towards a Classification Framework. In SEKE (pp. 225-229).

[9] Lenz, A. R., Pozo, A., & Vergilio, S. R. (2013). Linking software testing results with a machine learning approach. Engineering Applications of Artificial Intelligence, 26(5-6), 1631-1640.

[10] Braiek, H. B., & Khomh, F. (2020). On testing machine learning programs. Journal of Systems and Software, 164, 110542.

[11] Marijan, D., Gotlieb, A., & Ahuja, M. K. (2019, April). Challenges of testing machine learning based systems. In 2019 IEEE international conference on artificial intelligence testing (AITest) (pp. 101-102). IEEE.

[12] Kahles, J., Törrönen, J., Huuhtanen, T., & Jung, A. (2019, April). Automating root cause analysis via machine learning in agile software testing environments. In 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST) (pp. 379-390). IEEE.

[13] Lachmann, R. (2018, June). Machine learning-driven test case prioritization approaches for black-box software testing. In The European test and telemetry conference, Nuremberg, Germany.

[14] Nakajima, S. (2018, October). Quality assurance of machine learning software. In 2018 IEEE 7th Global Conference on Consumer Electronics (GCCE) (pp. 601-604). IEEE.

[15] Gove, R., & Faytong, J. (2012). Machine learning and event-based software testing: classifiers for identifying infeasible GUI event sequences. In Advances in Computers (Vol. 86, pp. 109-135). Elsevier.

[16] Zhang, D. (2006, November). Machine learning in value-based software test data generation. In 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06) (pp. 732-736). IEEE.

[17] e Silva, D. G., Jino, M., & De Abreu, B. T. (2010, April). Machine learning methods and asymmetric cost function to estimate execution effort of software testing. In 2010 Third International Conference on Software Testing, Verification and Validation (pp. 275-284). IEEE.

[18] Huang, S., Liu, E. H., Hui, Z. W., Tang, S. Q., & Zhang, S. J. (2018). Challenges of testing machine learning applications. International Journal of Performability Engineering, 14(6), 1275.

[19] Masuda, S., Ono, K., Yasue, T., & Hosokawa, N. (2018, April). A survey of software quality for machine learning applications. In 2018 IEEE International conference on software testing, verification and validation workshops (ICSTW) (pp. 279-284). IEEE.

[20] Li, J. J., Ulrich, A., Bai, X., & Bertolino, A. (2020). Advances in test automation for software with special focus on artificial intelligence and machine learning. Software Quality Journal, 28(1), 245-248.