Uluslararası İleri Doğa Bilimleri ve Mühendislik Araştırmaları Dergisi
Sayı 8, S. 201-220, 4, 2024

Araştırma Makalesi

International Journal of Advanced Natural Sciences and Engineering Researches
Volume 8, pp. 201-220, 4, 2024

Research Article

https://as-proceeding.com/index.php/ijanser
ISSN: 2980-0811

Measure The Software Quality Based On Bat Optimization Algorithm

Ahmed F. Rashad [*] and Shahla U. Umar [2]

[1,2] *Department of Computer Science, College of Computer Science and Information Technology, University of Kirkuk, Kirkuk, Iraq, University of Kirkuk, Kirkuk, Iraq*

** Corresponding author's Email: Ahmed.Fadil.rashad@gmail.com*

(Received:16 May 2024, Accepted: 25 May 2024)

(3rd International Conference on Engineering, Natural and Social Sciences ICENSOS 2024, May 16-17, 2024)

ATIF/REFERENCE: Rashad, A. F. & Umar, S. U. (2024). Measure The Software Quality Based On Bat Optimization Algorithm. *International Journal of Advanced Natural Sciences and Engineering Researches*, 8(4), 201-220.

Abstract – Measuring software quality is essential for software development as it affects the user experience and performance of software. Traditional methods of measuring software quality can be time-consuming and resource-intensive Therefore, the paper proposes a novel method based on bat optimization algorithm to measuring software quality. It is an optimization method inspired by nature and based on bats' echolocation behavior. The experiments on a data set of jm1 software projects that the bat optimization algorithm can effectively measure software quality. Regarding accuracy, the findings show that Decision Tree and Random Forest regularly beat the other classifiers. These models have excellent accuracy rates, suggesting their ability to properly categories software instances and identify possible quality concerns. KNN perform well, whereas the Multilayered Perceptron model and Adaboost performs poorly. Out of fiver classifier the performance of Decision tree and Random forest classifier is good, achieve Decision tree 99.7% and Random forest classifier 97.9% training accuracy.

Keywords – Machine Learning, Bat Optimization, Software Quality, Evaluation Matrix

I. INTRODUCTION

Software quality (SQ) is an essential component of software development projects carried out in accordance with established software standards and formats. often software developers have a tendency to disregard or vary from established procedures in the absence of external supervision, it is important to guarantee that the process is carried out via independent SQA activities such as project reviews and reviews by SQA employees [1][2]. The most effective instrument for software optimization in businesses is software quality. Since the objective is to count and give optimization, efficiency, and demand fulfillment, software must meet certain criteria to ensure its quality. In response to this need, a number of organizations or researchers have put forth methodologies, guidelines, and quality rules for creating and/or using software products and enable the evaluation of their quality throughout their life cycles, thereby fostering a quality environment built on effective information management [1, 2]. Software quality (SQ) may be characterized as a feature of a product that satisfies strict functional and performance standards, as well as certain development criteria and inherent features necessary for professionally developed software [3]. The principles and methods of software quality control is currently the subject of extensive discussion among academic and corporate organizations that have proposed a number of procedures and techniques for developing software quality norms [4][5]. Artificial intelligence (AI) techniques, which combine concepts from two different domains, are most widely used in software engineering validation processes. One of AI

domains, swarm optimization, is a superior option to find the best answer in the research area. It has comprehensive research capabilities and greatly improves the capacity to discover effective solutions within a reasonable period of time [6]. Meta-heuristic algorithms have two fundamental parts: exploitation and exploration. When searching for the global optimum during exploration, different solutions are investigated, however when searching for it during exploitation, the best recently found solutions are employed as the basis for the local search. [7][8][9]. Young developed the Bat method, a new meta-heuristic method for a variety of optimization problems, in [10]. This approach is based on how micro bats use echolocation. These creatures have developed echolocation as a hunting tool, and also have a remarkable guidance mechanism that allows them to distinguish even in complete darkness, between prey and other obstacles [11]. A completely new optimization technique the Bat Optimization Algorithm (BOA) is employed in this work. This method is used to check the quality of the software to the effectiveness in work and evaluation. The proposed approach uses a public dataset from the NASA database, which includes problem, benchmark and product data. This data's primary use is to offer project information of software programs. our proposed method, grounded in the Bat Optimization Algorithm, introduces a paradigm shift in software quality assessment. The dynamic adaptability, multi-objective evaluation, and user-centric features collectively position our approach as a pioneering solution, promising enhanced accuracy, comprehensive insights, and a more collaborative approach to measuring software quality in contemporary software engineering practices. Reminder of paper is constructing as fellow, section 2 given related work. In section 3 Material and Methods. In section 3 the proposed algorithm and describe the bat optimization algorithm, in section 4 describe expression result, in section 5 describe Discussion and final section conclusion.

II. RELATED WORK

Bestoun S. ahmed et al. [12] developed the "particle swarm test generator PSTG " approach in [2012] for evaluating software quality. This method uses appropriate case and PSO to build improved test sizes for the majority of configurations. The study also assesses and illustrates PSTG's capacity to produce effective test suites. To increase the predictability of software defects, Particle swarm optimization and the bagging method algorithm are two artificial algorithms that Romi Satria Wahono et al [13] used in 2013 to analyze the quality of products using NASA criteria. The packing approach is utilized to address the imbalance issue when PSO is employed for the chosen features. Wang et al.'s [14] application of ensemble feature selection approaches on 16 software defect datasets led them to conclude that ensembles with only a few rankers are more successful than ensembles with many or all rankers. The class imbalance issue has an impact in several areas, including software defect prediction. In the literature, augmentation, pooling, and sampling of data have all been proposed as solutions to class imbalance. Data sampling, which includes balancing the relative class distributions in the current data set, is the major tactic for dealing with class imbalance. Both oversampling and under sampling are valid methods for data sampling [15]. In the [2015] study of Deepti Arora et al. [16], By figuring out where mistakes are most likely to occur, two artificial methods—particle swarm optimization (PSO) and genetics—were merged for software testing. A researcher's comparison between these two approaches generally makes it impossible to test all potential inputs or detect errors because the software is not fully tested. Rajesh Kumar Sahu1 et al. [17] in [2016] proposed a technique to integrate three artificial methods generated by the heuristics of (harmony search, particle swarm optimization and bee colony optimization) to discover the best test mode for the data. The researchers used these strategies to create better test cases, using an ATM that wasn't working as an example and a random test case to find the optimum solution. This study by Kumar et al. [18] proposed a method for evaluate software quality. This work shows optimization results for software testing, they utilized two optimization techniques, Cuckoo Search Algorithm (ICSA) and PSO, with the goal of minimizing execution time, effort, and cost and increasing project quality and productivity. Ahmed et al. [19] utilized

multi-objective particle swarm optimization (PSO), a method that considers input parameters and the properties of every configuration, to find the optimal test suites. This approach produces an optimal combinatorial test suite that satisfies constraints between input properties. This paper by Gheisari et al. [20] developed a mathematical model based on factors (Q) to forecast customer satisfaction. This proposed utilized the relationship effects of the separation sides in mathematical optimization models to validate the real data. This approach offers a speed mechanism while still finding several answers in an important area. Models are based on the highest and lowest Q values. Constraints are what make high-quality software features. The Grasshopper Optimization Algorithm (GOA) is employed in [2021] Ibrahim Ahmed Saleh et al [21] to enhance software quality. The quality requirements that were used to test the certified software were determined using a variety of quality metrics. software defect is the main focus of software testing. This study also introduces GOA, which extracts the best features for testing and evaluating a group of software applications achieved results (%99.2) in 500 iterations. The particle swarm optimization (PSO) algorithm was used in [2018] Ibrahim Ahmed Saleh et al. [22] to measure software quality. The proposed algorithm was applied using (a2011R Matlab) based on validity criteria that include calculating classification accuracy, calculating the error percentage, the mean square error, and the root mean square error. They achieved results (%98) in 500 iterations. NASA standards data were used in the article. The experiment and results demonstrate that the grasshopper optimization technique, which is based on feature selection and bagging for all classification methods used in research, has enhanced performance quality to Forecast of software flaws. In Table 1 present summary of related work.

Table 1: Summary of Related Work

Ref.	PSTG	PSO	bagging method	genetics	harmony search	bee colony optimization	GOA	multi-objective (PSO)	mathematical model based on factors	ICSA
[12]	√	×	×	×	×	×	×	×	×	×
[13]	×	√	√	×	×	×	×	×	×	×
[16]	×	√	×	√	×	×	×	×	×	×
[17]	×	√	×	×	√	√	×	×	×	×
[18]	×	√	×	×	×	×	×	×	×	√
[19]	×	×	×	×	×	×	×	√	×	×
[20]	×	×	×	×	×	×	×	×	√	×
[21]	×	×	×	×	×	×	√	×	×	×

III. MATERIAL AND METHODS

Using a mix of the bat optimization technique and machine learning classification algorithms, the research article seeks to evaluate the quality of software. The researchers used a variety of pre-processing approaches to clean up the raw data and convert it into a format that the machine learning algorithms could understand in order to accomplish this aim. Data normalization, feature scaling, feature selection, and data cleaning were some of the pre-processing methods used. Following that, a number of performance criteria, including accuracy, precision, recall, F1 score, and area under the receiver operating characteristic (AUC) curve, were used to assess how successful the suggested technique was. Software quality monitoring

revealed encouraging results when the bat optimization algorithm and machine learning classification algorithms were used with the suitable pre-processing methods.

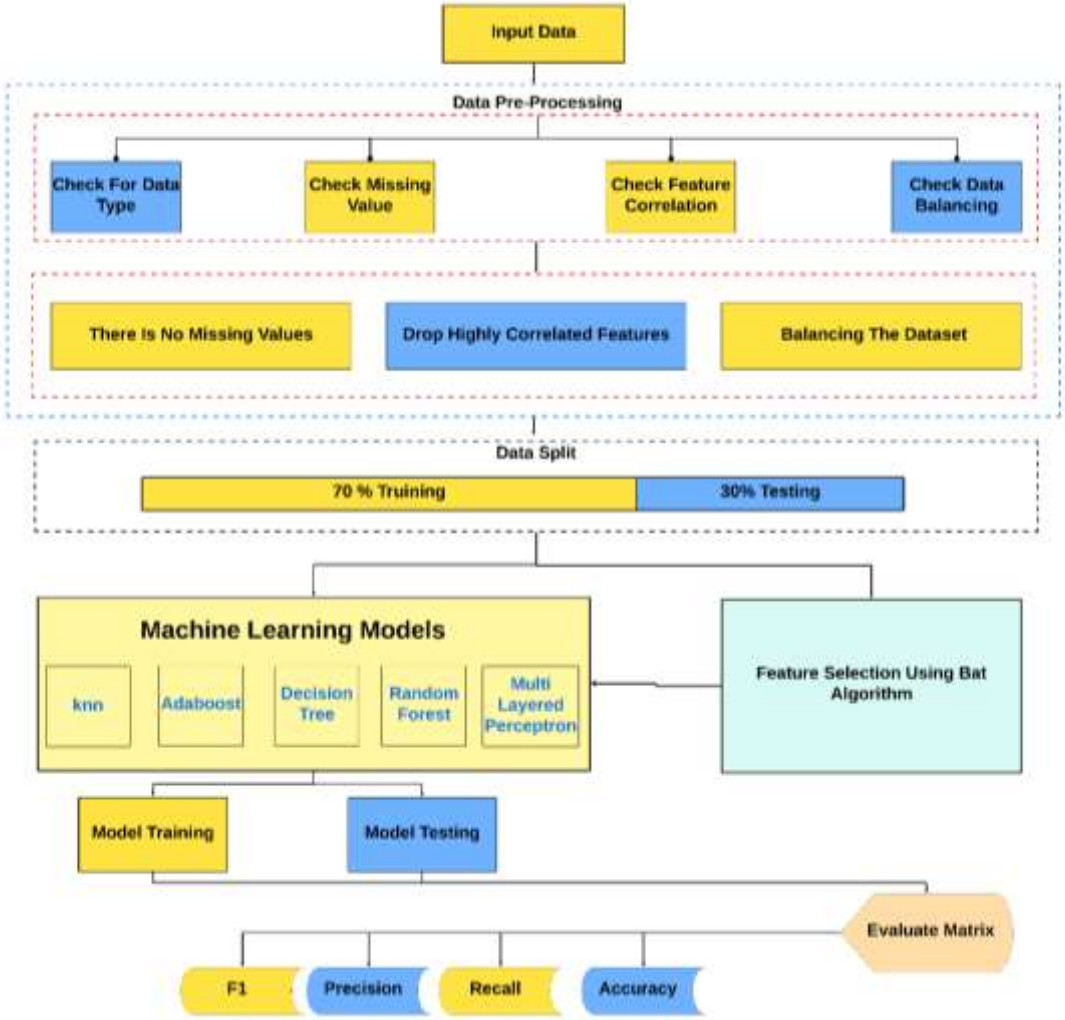

Figure 1: Workflow of Proposed Methodology

I. Datasets

The JM1 data set is a collection of software metrics for over 7000 Java modules, which are divided into two classes: defective and non-defective. The data set contains 21 attributes, such as lines of code, McCabe's complexity metrics, and Halstead's software metrics, among others. The dataset is often used in research on software defect prediction to evaluate the accuracy with which machine learning approaches discover faulty modules.

II. Data analysis and pre-processing

In data analysis and pre-processing, we first examine the data set for missing values, outliers, and imbalanced classes. We also perform data normalization to ensure that all features have a similar range of values. we use techniques such as oversampling to address class imbalance in the data set. Then to assess the effectiveness of the machine learning models, we divided the data set into training and testing sets. Finally, we also perform feature selection to identify the most relevant features for classification.

III. Bat optimization algorithm

Yang, a researcher, developed the Bat Algorithm in 2010 to address various optimization problems. The method is developed and used to improve the fitness function in a manner that is similar to the natural behaviour of bats. Yang presented a number of guidelines in 2010 [22] to mimic the behaviour of the bat's echolocation system and utilize them for construct this algorithm. Bats use their abilities to detect

echolocation to measure distance. They can distinguish between insects and background obstructions. Bats always fly randomly when hunting prey at a certain velocity (vi) at a location (xi) with a fixed frequency (f_min) but variable wavelength (λ) and loudness (A0). The volume of a bat's pulse can vary in several ways. Yang, however, believed that it varied between a big positive value (A0) and a small constant number (Amin). In the simulation, the frequency, loudness, and pulse rate were tuned to create a new solution (Equations (1)– (3)). The new solution is compared to the old one based on how close to the ideal solution it is, which determines how finely the solutions modified by the loudness and pulse rate are.

$$x_{it+1} = x_{it} + v_{it+1} \quad (1)$$

where (xit) denotes a bat's location, (vit+1) denotes its velocity, and (xit+1) is a bat's updated position at time t+1. The following formula is used to get bat I's velocity at time t+1:

$$v_{it+1} = v_{it} + A^*(x_{best} - x_{it}) + r^*(x_{jt} - x_{it}) \quad (2)$$

where (x best) is the best solution that has yet to be found by any bat, r is a chance number between -1 and 1, and A is the bat's loudness. (x jt) is the position of another randomly selected bat at time t. the BOA also uses a frequency scaling factor, f_min, in order to manage the search's step size. The step size decreases as the algorithm progresses, which helps to refine the search around promising solutions. bat algorithm is A strong optimization method that may be used to address a variety of optimization issues. Numerous real-world issues have been effectively addressed using it in a variety of industries, including engineering, healthcare, and banking [23].

IV. Classification via Machin learning

We do in this chapter classify the software quality based on the JM1 data set using five different machine learning algorithms: RandomForest Classifier, AdaBoost Classifier, KNN, Decision Tree, and multilayer perceptron. The pre-processed data set will be used to train each of these algorithms, and different metrics including accuracy, precision, recall, and F1-score will be used to assess each algorithm's performance. We want to determine which classifier produces the best predictions of software quality based on the provided data set by comparing the outcomes of different classifiers.

D.1 Random forest classifier

The Random Forest Classifier is an ensemble learning technique for classifying data that generates the class that is the mode of the classes (classification) or mean prediction (regression) of the individual decision trees. Random Forest is an algorithm can handle large amounts of data with high dimensionality, can avoid overfitting, and can provide feature importance ranking. The algorithm can be explained mathematically as follows: randomly select N samples from the training set with replacement.

- Construct a decision tree using the selected N samples.
- Repeat steps 1 and 2 to create k decision trees.
- For each new data point, predict the class by having it pass through each decision tree in the forest and taking the mean for regression, or the mode of the predicted classes.
- Output the predicted class.

The RandomForest Classifier uses the following two methods to reduce overfitting:

- Bagging: Bootstrap Aggregating (or Bagging) is a method that helps to reduce variance and overfitting by randomly selecting a subset of the training set with replacement to train each decision tree.
- Random Feature Selection: When breaking up each node in the decision tree, the algorithm chooses a random subset of the characteristics to take into account. This promotes more decision tree variety and avoid overfitting to specific features.

- An effective machine learning technique for classification problems is the Random Forest Classifier, particularly when working with big and high-dimensional datasets. [26].

D.2 AdaBoost classifier

AdaBoost (Adaptive Boosting) is an ensemble learning method an ensemble learning technique is called AdaBoost (Adaptive Boosting). in which several weak learners are combined to create a strong learner. In this algorithm, a base classifier is trained on the entire data set, and then additional classifiers are sequentially trained on the examples that the previous classifiers have not classified correctly. Each classifier has a weight, and the weights are updated based on the classification error of the previous classifier. Each classifier's prediction is integrated and weighted according to its unique accuracy to get the final prediction. Mathematically, the AdaBoost algorithm can be represented as follows: Set all training example weights to 1/n, where n is the total number of training examples.

$$w\_i = 1/n, \text{ for } i = 1, 2, n \quad (3)$$

for each iteration t = 1,2,...,T do the following: Train a weak classifier h_t on the training data with weights w_i. Compute the weighted error ε_t of the weak classifier h_t.

$$\varepsilon\_t = \Sigma\_i\, w\_i * (y\_i \neq h\_t(x\_i)) / \Sigma\_i\, w\_i \quad (4)$$

compute the weight α_t of the weak classifier h_t.

$$\alpha\_t = \log((1-\varepsilon\_t) / \varepsilon\_t) \quad (5)$$

update the weights of the training examples.

$$w\_i = w\_i * \exp(\alpha\_t * (y\_i \neq h\_t(x\_i))), \text{ for } i = 1,2,\ldots,n$$

normalize the weights so that they sum to 1.

$$w\_i = w\_i / \Sigma\_i\, w\_i, \text{ for } i = 1,2,\ldots,n \quad (6)$$

create the final classifier by combining the weak classifiers in a weighted fashion.

$$H(x) = \text{sign}(\Sigma\_t\, \alpha\_t * h\_t(x)) \quad (7)$$

Here, y_i is the corresponding class label and x_i is the i-th training case, and h_t(x_i) is the prediction of the weak classifier h_t for the example x_i. The sign function returns +1 if the weighted sum is positive and -1 otherwise, and is used to make the final prediction for a given example x[25], [27].

D.3 KNN

- It is a Machine learning algorithms used for classification and regression include K-Nearest Neighbours (KNN). data point's categorization in KNN is based on the feature space's k-nearest neighbours' dominant class. A distance metric, like the Euclidean or Manhattan distance, is used to calculate the distance between data points. The KNN algorithm may be summarized using the phases listed below:
- select the number of neighbours (k).
- Determine the distances between each new data point and every existing data point in the data set. Then, based on those distances, choose the k neighbors who are closest to the new data point.

- After that selected the new discovered data point to the most common class between the k nearest neighbour's.

The equation for computing distance between two data points x and y is:

Distance (x, y) = sqrt (sum((x_i - y_i)2) for i in range(n))

As shows in equation the distance between two data points, x and y. Where n is the number of features in the data set and x_i and y_i are the values of the ith feature for the corresponding x and y data points.The following equation have been utilized to calculate Manhattan distance, which is another useful metric in KNN:

Sum (abs(x_i-y_i)) for i in range(n) equals distance (x, y).

This method is simple and effective, its performance may be greatly impacted by the value of k. While a high value might result in underfitting, a low quantity of k could lead to overfitting. Therefore, to determine k, cross-validation or other methods should be used [28].

D.4 Decision tree

This model is a type of a supervised learning technique for regression analysis and classification. This approach utilized to creates a tree-like representation of choices and their results. In this model the tree is building until the subsets are as pure as is practical with recognize to the target variable, the dataset is sometimes separated into smaller subsets dependent on the most significant feature. In decision tree classification, the algorithm selects the root node by giving a score to each attribute, after which it creates sub-nodes according to the chosen attribute. Typically, a metric like entropy or Gini impurity is used to calculate the score. The Gini impurity measures the probability that a random sample will be incorrectly categorized, while entropy measures the degree of data disorder. The Gini impurity formula is the fraction of samples in class node. The decision tree technique stops splitting the subgroups into smaller subsets based on the same measure when a stopping condition is fitted, such as the maximum depth of the tree or the minimum number of samples at a leaf node. Once the tree is formed, it can be utilized for classification by running it through the feature values of a new sample and giving the value of every node to the leaf node that links to the predicted class. [26].

D.5 Multi-layer perceptron

This model can be utilized on regression and classification. The MLP contain from three layers as an input layer, one or more hidden layers, and an output layer. As shows on the following equations.

$$\$z^h = f(W^h x + b^h) \$ \quad (11)$$

$$\$a^h = g(z^h) \$ \quad (12)$$

Where x is the input data, the weights and bias of the hidden layer are shown by W^h and b^h respectively. The hidden layer's activation function is refer to the symbols h, f, and the weighted sum of the input x, the weights W^h, and the bias, z^h. The hidden layer's output activation function is the symbols$g and b^h with reference to the output layer:

$$z^o = f(W^o a^{h_n} + b^o) \quad (13)$$

$$a^o = g(z^o) \quad (14)$$

Where the a^{h_n} is the output of the final hidden layer, while f and g are the activation functions for the output layer, W^o and b^o are the output layer's weights and bias. The backpropagation method is utilized to train the MLP in order to minimize a loss function, such as cross-entropy or mean squared error. During training, the weights and biases are corrected to minimize the difference between the expected and actual outputs [26], [27].

V. RESULTS

This section shown the data collection and the results analysis to evaluate findings hypotheses.

A. Data analysis and pre-processing

There are 22 columns and 7782 rows in this dataset. For every column in the data collection, the count, mean, standard deviation, minimum, and maximum values are given. The data distribution may be comprehended by using the average and standard deviation of every column. For instance, the mean and standard deviation of the CYCLOMATIC COMPLEXITY column are 6.78 and 14.08, respectively, suggesting that the data is positively skewed and that there are likely some outliers in the data. There are two possible values for the Defective variable, and the counts of each value are shown in the output. The first value is 0, which appears 6110 times. The second value is 1, which appears 1672 times. It is possible that this data represents the number of defective items in a manufacturing process, with 0 indicating a non-defective item and 1 indicating a defective item. Figure 2 shows that the data set is imbalanced.

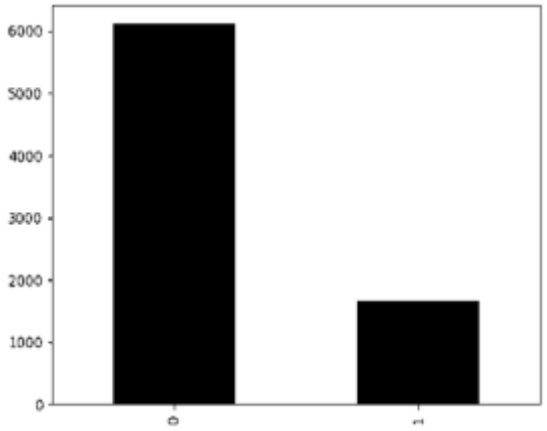

Figure 2: Distribution of class variable

B. Random oversampling

After oversampling, the data set with 7782 observations and 21 features. The second number in parentheses represents the target variable, which also has a shape of (7782,) before balancing classes. After balancing the classes, your data set had a shape of (12220, 21) and a target variable with a shape of (12220,).

C. Dataset split

The train_test_split function is used to split the data set into two sets - training set and testing set. In this

case, the function is splitting the X (input features) and y (target variable) data into X_train, X_test, y_train, and y_test datasets with the test size option set to 0.3, 30% of the data will be utilized for training and the remaining 70% for testing. Every time the code is run, the same split is produced since the random_state argument is set to 42.

D. Machine learning models on all dataset

D.1 Training comparison explanation

Performance indicators are given for the following machine learning algorithms in the table: KNN, Adaboost, Decision Tree, Random Forest, and Multi-Layer Perceptron. Accuracy, Recall, Precision, and F1 score are the four-assessment metrics used to assess each algorithm's performance.The ratio of instances successfully predicted to all instances, or the accuracy measures the overall accuracy of the algorithm's predictions. In this situation, the Decision Tree, Random Forest, and Multi-Layered Perceptron algorithms all had high accuracy ratings of 0.998, demonstrating that they correctly predicted almost all instances in the data set. KNN and Adaboost scored slightly less accurate than the other algorithms (0.805 and 0.659, respectively). It was evident that their predictions often needed to be more accurate. Recall gauges an algorithm's accuracy in correctly identifying every positive instance. The ratio of accurate optimistic predictions to the total of real positives and accurate negatives is computed. The Multi-Layered Perceptron, Decision Tree, and Random Forest algorithms achieved perfect recall scores of 1.00, accurately identifying all positive events. Recall ratings for KNN and Adaboost were lower, at 0.897 and 0.604, respectively, indicating that some cases were overlooked. Precision measures how well an algorithm can identify positive occurrences among all the projected positive cases. The ratio of accurate forecasts to the total of accurate predictions and false positives is computed. The Multi-Layered Perceptron, Decision Tree, and Random Forest algorithms all received excellent precision scores of 0.996, indicating few incorrect optimistic predictions. KNN and Adaboost both had lower precision scores (0.756 and 0.678, respectively), which would indicate that there were false positives. Precision and recall are combined into a single metric called the F1 score, which balances both parameters. It is figured out as the harmonic mean of recall and precision. High F1 scores of 0.998 were attained by the Decision Tree, Random Forest, and Multi-Layered Perceptron algorithms, demonstrating an excellent balance between recall and precision. Lower F1 scores from KNN and Adaboost—0.821 and 0.639, respectively—indicate a trade-off between recall and precision. All evaluation measures showed that the Decision Tree, Random Forest, and Multi-Layered Perceptron algorithms worked well, with excellent accuracy, recall, precision, and F1 scores. The performance of KNN and Adaboost could have been better, with poorer accuracy, recall, precision, and F1 scores.

Table 2: Training comparison explanation

Models	Accuracy	Recall	Precision	F1
KNN	0.805	0.897	0.756	0.821
Adaboost	0.659	0.604	0.678	0.639
Decision Tree	0.998	1	0.996	0.998
Random Forest	0.998	1	0.996	0.998
Multi Layered Perceptron	0.630	0.549	0.654	0.597

D.2 Testing comparison explanation

Five machine learning algorithms' performance statistics are included in the table: KNN, Adaboost, Decision Tree, Random Forest, and Multi-Layer Perceptron. Accuracy, Recall, Precision, and F1 score are the four metrics used to assess performance.
The overall accuracy of the algorithm's predictions is represented by accuracy. The Random Forest algorithm provided accurate predictions for almost 90.4% of the cases in this example, earning it the highest accuracy score of 0.903. Adaboost came in second with a score of 0.838, followed by KNN with 0.675, the Decision Tree algorithm with 0.838, and Multi-Layered Perceptron with the lowest accuracy score of 0.622. Recall gauges an algorithm's capacity to recognize positive events with accuracy. The Decision Tree algorithm detected roughly 94.1% of the positive examples, earning it the highest recall score of 0.940. The recall score for Random Forest was 0.941, while the recall scores for KNN, Adaboost, and Multi-Layered Perceptron were 0.768, 0.599, and 0.551, respectively. Precision measures how well an algorithm can identify positive cases among those projected to be positive. The Decision Tree algorithm correctly recognized roughly 78.2% of the positive occurrences, earning it the highest precision score of 0.782. While KNN, Adaboost, and Multi-Layered Perceptron all had lower precision ratings of 0.650, 0.666, and 0.646, respectively, Random Forest had a significantly lower precision score of 0.876. Precision and recall are combined into a single metric called the F1 score, which balances both parameters. The Decision Tree method, with an F1 score of 0.854, has the highest precision-to-recall ratio. While KNN, Adaboost, and Multi-Layered Perceptron all had lower F1 values of 0.704, 0.630, and 0.595, Random Forest had a little lower F1 score of 0.908. All evaluation measures showed that the Decision Tree and Random Forest algorithms performed effectively, with reasonably high accuracy, recall, precision, and F1 scores. The performance of KNN and Adaboost could have been better, with poorer accuracy, recall, precision, and F1 scores. The performance of the Multi-Layered Perceptron method was the worst of the five algorithms.

Table 3: Testing comparison explanation

Models	Accuracy	Recall	Precision	F1
KNN	0.675	0.768	0.650	0.704
Adaboost	0.647	0.599	0.666	0.630
Decision Tree	0.838	0.940	0.782	0.854
Random Forest	0.903	0.941	0.876	0.908
Multi Layered Perceptron	0.622	0.551	0.646	0.595

D.3 Confusion matrix for best model

1. Decision tree

In Figure 3 The presented confusion matrix illustrates how well a Decision Tree algorithm performs in assessing software quality based on the bat optimization approach when Training a model. The matrix is divided into four quadrants, each of which representing a different classification conclusion. True positive (TP) predictions are shown in the top-left quadrant, which is emphasized as 49.95%.it indicates that the algorithm successfully recognized high software quality while measuring software quality. The high value in this quadrant denotes a sizable proportion of correctly predicted good outcomes. The bottom-right quadrant represents the actual negative (TN) forecasts, marked as 49.85%. It indicates that the algorithm

properly recognized instances of subpar software quality in measuring software quality. This quadrant's high score denotes a sizable proportion of accurate pessimistic predictions. Minimal percentages are present in the top-right and bottom-left quadrants, designated as 0.20 and 0.00%, respectively. These represent the false positive (FP) and false negative (FN) predictions. (FP) in software quality measurement is when the algorithm mistakenly identifies a case as having good software quality when it had low software quality. On the other hand, (FN) is when the algorithm incorrectly categorizes an instance as having bad software quality when it has acceptable quality. The Decision Tree algorithm correctly detected cases of both good and bad software quality, as seen by the high percentages in (TP) and (TN) quadrants. However, given that miscommunications might result in inaccurate software quality assessments, it is crucial to look into and address the low percentages in the false positive and false negative quadrants.

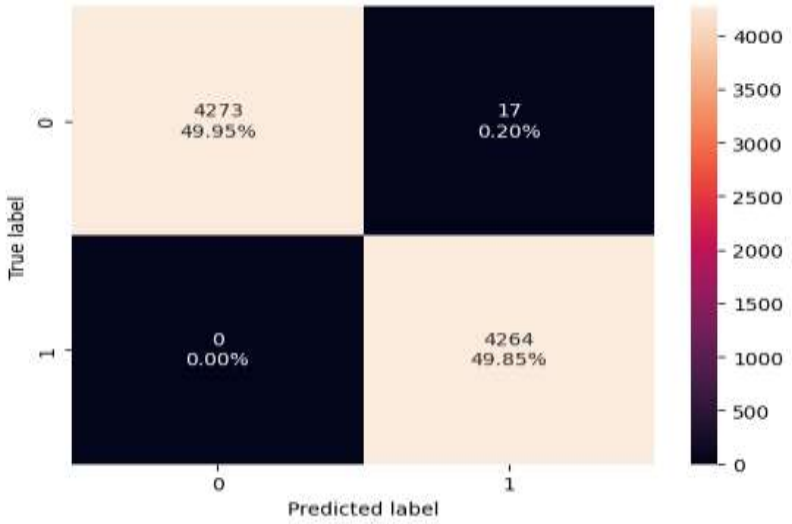

Figure 3: Confusion matrix of Training model for Decision Tree

In Figure 4 The performance of a Decision Tree method when testing a model is represented by the provided confusion matrix. quadrants the matrix each representing a different classification conclusion. (TP) predictions are shown in the top-left quadrant, which is 36.47%. The Top-right quadrant represents (FP) predictions, which is marked as 13.18%. (FN) predictions are shown in the bottom-left quadrant, highlighted as 2.97%. (TN) forecasts are shown as 47.38% in the bottom-right quadrant. In the context of model testing, the decision tree algorithm did a fair job of correctly recognizing positive and negative cases, as seen by the percentages in the actual positive and negative quadrants. However, the percentages in the false positive and false negative quadrants suggest that some miscommunications may have occurred. In order to improve the model's performance and accuracy, it is crucial to look into and correct these miscommunications.

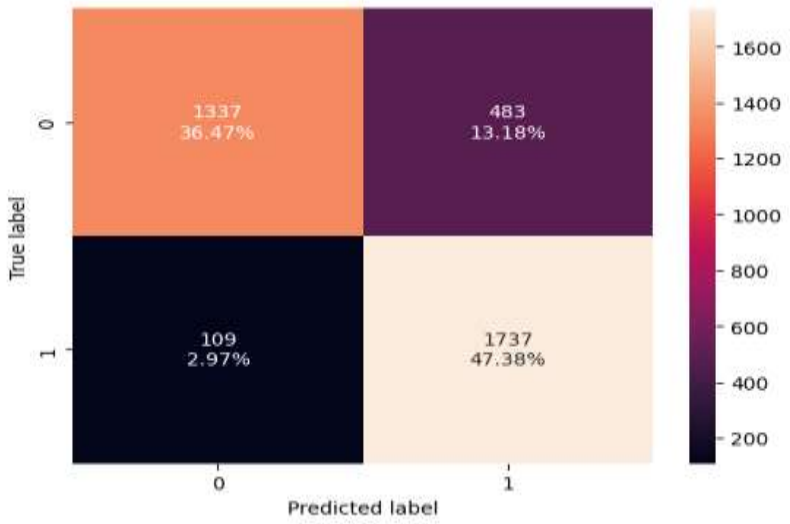

Figure 4: Confusion matrix of Testing model for Decision Tree

2. Random forest

In Figure 5 The provided confusion matrix is a representation of how a Random Forest model performed during the training phase. The top-left quadrant of the graph displays (TP) forecasts, which are highlighted as 49.95%. Forecasts for (TN) are shown in the bottom-right quadrant and are designated as 49.85%. Tiny percentages are in the top-right and bottom-left quadrants, designated as 0.20 and 0.00%, respectively. These represent the (FP) and (FN) predictions. In this instance, extremely few erroneous positive and false pessimistic predictions were made by the Random Forest model during training. The Random Forest model did well in properly recognize occurrences of both positive and negative classes throughout the training phase, as evidenced by the high percentages in the true positive and true negative quadrants. According to the low percentages in the false positive and false negative quadrants, the model did not make many incorrect classifications. It demonstrates the Random Forest model's high training performance, with precise predictions for both positive and negative examples. The performance of a Random Forest model on test data is represented by the confusion matrix that is delivered. In Confusion matrix of testing model for Random forest model (Figure 6) explanation (TP) predictions are 42.99% of the top-left quadrant. The percentage of events correctly identified as positive is represented by the value in this quadrant. The top-right quadrant represents (FP) predictions, which are 6.66%. (FN) predictions are shown in the bottom-left quadrant, highlighted as 2.95%. (TN) predictions are shown as 47.41% in the bottom-right quadrant. The Random Forest model did a fair job of correctly recognizing positive and negative cases throughout the testing phase, according to the percentages in the correct positive and negative quadrants. However, as seen by the percentages in the false positive and false negative quadrants, there were some miscommunications. For the model to be more accurate and perform better on test data that has yet to be seen, it is crucial to look into and fix these miscommunications.

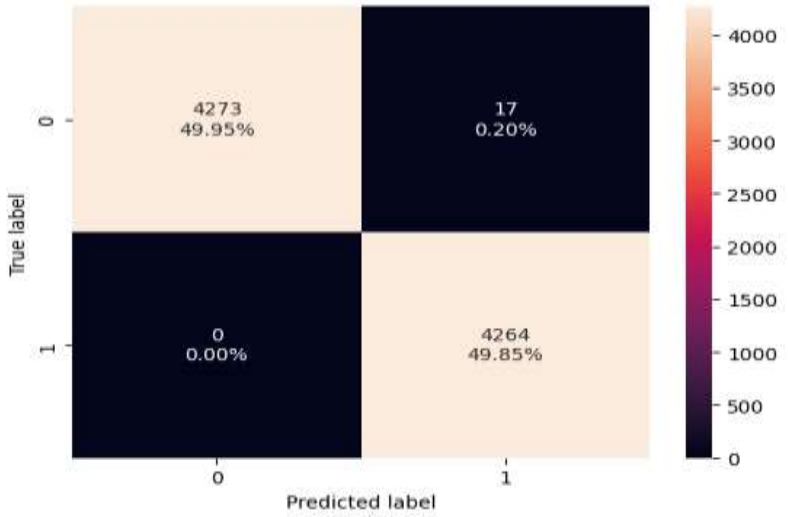

Figure 5: Confusion matrix of Training model for Random Forest model explanation

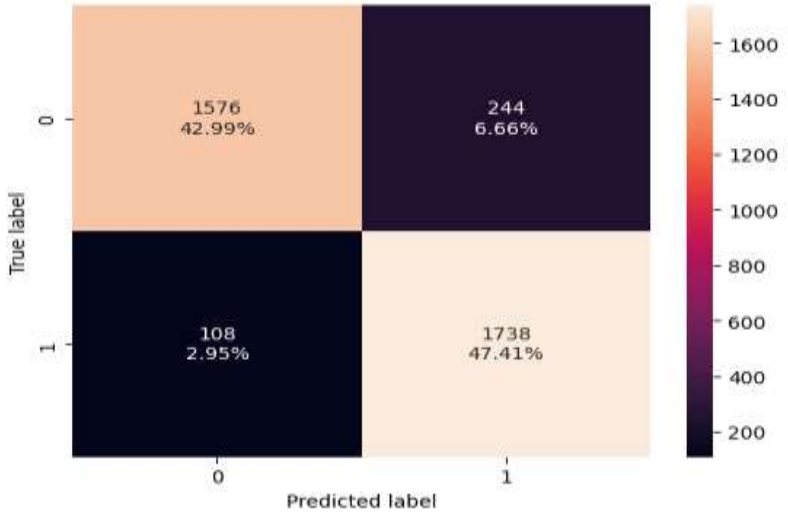

Figure 6: Confusion matrix of testing model for Random forest model explanation

C. Machine learning models on selected features using the bat algorithm

The following are the findings of assessing software quality using the BAT optimization algorithm, where N= number of features selected, N =random number between (1-21):

Table 4: Number of features selected

Models	Number of features selected as randomly
Decision Tree	10
Random Forest	3
KNN	1
Adaboost	8
Multi Layered Perceptron	5

C.1 Training comparison explanation

The following are the findings of assessing software quality using the BAT optimization algorithm in training phase: The accuracy of KNN was 0.788, suggesting that it successfully categorized 78.8% of the occurrences. With a recall score of 0.877, it effectively recognized 87.7% of positive cases. The accuracy rating 0.744 implies that 74.4% of the positive cases were genuine positives. The F1 score was 0.805, which combines accuracy and recall. The accuracy of AdaBoost was 0.642, meaning that it successfully categorized 64.2% of the cases. With a recall score of 0.564, it effectively recognized 56.4% of positive cases. The accuracy rating of 0.666 implies that 66.6% of the positive cases were true positives. F1's score was 0.611. The decision Tree worked well, with an accuracy of 0.997, suggesting that 99.7% of the occurrences were identified correctly. Its recall score of 0.999 effectively detected 99.9% of actual positive cases. The accuracy rating 0.994 shows that 99.4% of the positive cases were genuine positives. F1's score was 0.997. Random Forest's accuracy was 0.979, successfully identifying 97.9% of the occurrences. With a recall score of 0.986, it effectively recognized 98.6% of actual positive instances. The accuracy rating 0.972shows that 97.2% of the positive cases were true positives. F1's score was 0.979. The accuracy of the Multilayered Perceptron model was 0.617, meaning that it correctly categorized just 61.7% of the occurrences. With just 56.7% of real positive cases captured, the recall score of 0.567 indicates it had trouble detecting actual positive instances. The accuracy rating of 0.628 implies that 62.8% of the positive events were genuine positives. The F1 result was 0.596. In summary, the BAT optimization method demonstrated diverse degrees of effectiveness for the various software quality models studied. The Decision Tree and Random Forest performed admirably, with high accuracy, recall, precision, and F1 ratings. KNN performed well, whereas the Multilayered Perceptron model and AdaBoost performed poorly, notably regarding the recall and F1 score.

Table 5: Training comparison explanation on selected features using the bat algorithm

Models	Accuracy	Recall	Precision	F1
KNN	0.788	0.877	0.744	0.805
Adaboost	0.642	0.564	0.666	0.611
Decision Tree	0.997	0.999	0.994	0.997
Random Forest	0.979	0.986	0.972	0.979
Multi Layered Perceptron	0.617	0.567	0.628	0.956

C.2 Testing comparison explanation

The following are the test results for the various classifiers: The accuracy of KNN was 0.674, suggesting that it successfully categorized 67.4% of the cases. With a recall score of 0.770, it effectively recognized 77% of positive cases. The accuracy rating 0.648 implies that 64.8% of the positive cases were genuine positives. The F1 score was 0.704, which combines accuracy and recall. The accuracy of AdaBoost was 0.630, suggesting that it properly categorized 63% of the cases. A recall score of 0.560 effectively determined 56% of positive cases. The accuracy rating of 0.655 implies that 65.5% of the positive events were genuine positives. F1's score was 0.603. The decision Tree worked well, with an accuracy of 0.852, suggesting that 85.2% of the occurrences were identified correctly. With a recall score of 0.953, it effectively recognized 95.3% of positive cases. The accuracy rating of 0.795 implies that 79.5% of the positive cases were genuine positives. F1's score was 0.867. Random Forest's accuracy was 0.859,

successfully categorizing 85.9% of the occurrences. With a recall score of 0.930, it effectively recognized 93% of positive cases. The accuracy rating of 0.816 implies that 81.6% of the positive cases were genuine positives. F1's score was 0.869. The accuracy of the Multilayered Perceptron model was 0.605, suggesting that it properly categorized just 60.5% of the occurrences. With just 56.7% of actual positive instances captured, the recall score of 0.567 indicates it had trouble detecting true positive cases. The accuracy rating of 0.617 implies that 61.7% of the positive cases were true positives. F1's score was 0.591. In summary, the Random Forest models and Decision Tree outperformed the other classifiers regarding accuracy, recall, precision, and F1. KNN performed well, whereas the Multilayered Perceptron model and AdaBoost performed poorly, notably regarding the recall and F1 score.

Table 6: testing comparison explanation on selected features using the bat algorithm

Models	Accuracy	Recall	Precision	F1
KNN	0.674	0.770	0.648	0.704
Adaboost	0.630	0.560	0.655	0.603
Decision Tree	0.852	0.953	0.795	0.867
Random Forest	0.859	0.930	0.816	0.869
Multi Layered Perceptron	0.605	0.567	0.617	0.591

C.3 Confusion matrix for best model

1. Decision tree

In Confusion matrix two classes in this case: 0 and 1. According to the confusion matrix, the model accurately predicted 4268 occurrences as class 0 and 4262 as class 1. These are the actual winners in their respective fields. Furthermore, the model predicted 22 cases as class 1 when they were, in fact, class 0 (false positives) and 2 instances as class 0 when they were, in fact, class 1 (false negatives). We may further understand the Decision Tree model's performance by analysing these values in accuracy, precision, recall, and F1 score measures. In this study utilised accuracy to evaluates the model's predictions, while precision focuses on the model's ability to identify true positives, and recall refer to the model's ability to detect false positives. The model's accuracy in this scenario indicates that it correctly detected the majority of occurrences. The model is a good in error reduction, as seen by the relatively low numbers of false positives (22) and false negatives (2). The equal totals of true positives in both classes further show the model's ability to shown accurate predictions for both class 0 and class 1. The confusion matrix of the Decision Tree model shows useful information about how well it performs in classification, enabling us to evaluate how well it can predict examples that belong to classes 0 and 1.

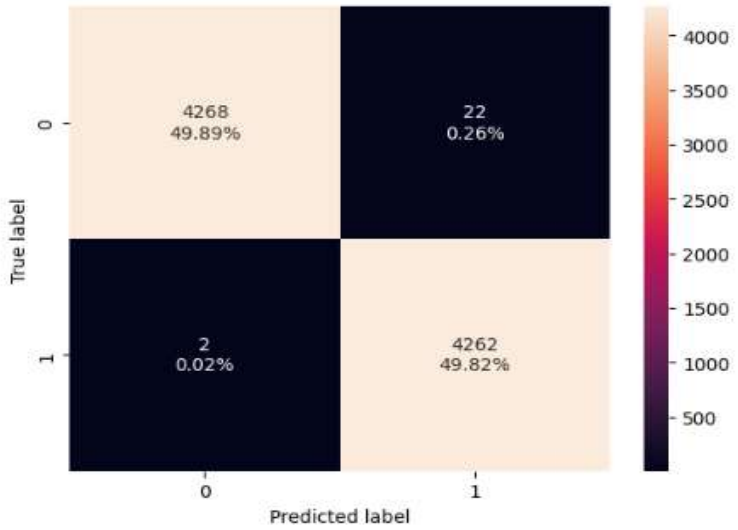

Figure 7: Confusion matrix of the Decision Tree training model using the bat method

In this case, there are two classes in the testing phase (0 and 1). The model correctly identified 1366 occurrences as class 0 and 1761 as class 1, according to the confusion matrix. In their respective disciplines, they are the real winners. Moreover, 454 examples of false positives and 85 cases of false negatives were predicted by the model as class 0 and class 1, respectively, when they were really class 0 (false positives and class 0). The comparatively low number of false positives (454) and false negatives (85) indicates that the model did well in error reduction. The model's capacity to produce correct predictions for both class 0 and class 1 is further demonstrated by the balanced levels of true positives in both classes.

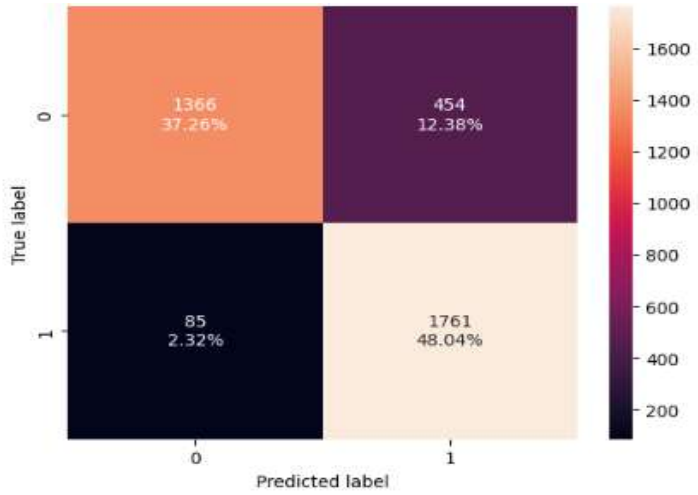

Figure 8: Confusion matrix of Testing model for Decision Tree using bat algorithm

2. Random forest

Confusion matrix in training phase content two classes in this case: 0 and 1. According to the confusion matrix, the model accurately predicted 4170 occurrences as class 0 and 4208 as class 1. These are the actual winners in their respective fields. Furthermore, the model predicted 120 cases as class 1 when they were, in fact, class 0 (false positives) and 56 instances as class 0 when they were, in fact, class 1 (false negatives). The comparatively low number of false positives (120) and false negatives (56) indicates that the model

did well in error reduction. The model's capacity to produce correct predictions for both class 0 and class 1 is further demonstrated by the balanced levels of true positives in both classes.

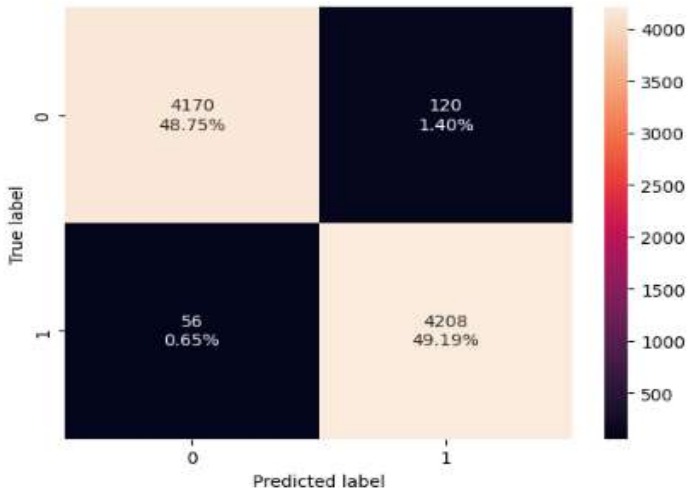

Figure 9: Confusion matrix of Training model for Random Forest using bat algorithm

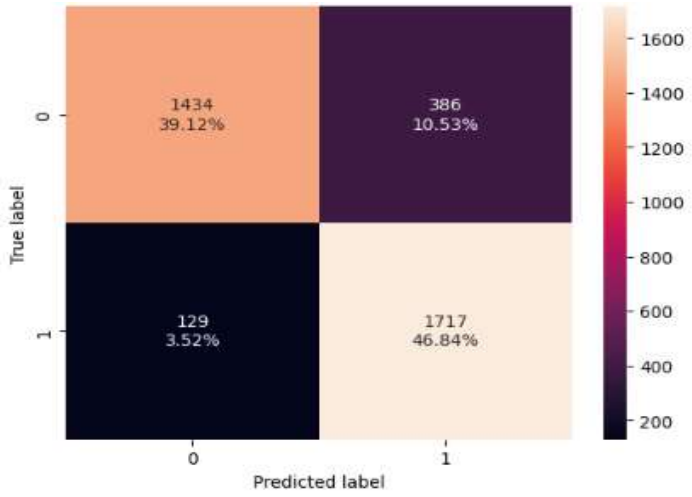

Figure 10: Confusion matrix of testing model for Random Forest using bat algorithm

Confusion matrix in training phase content two classes in this case: 0 and 1. According to the confusion matrix, the model accurately predicted 1434 occurrences as class 0 and 1717 as class 1. These are the actual winners in their respective fields. Furthermore, the model predicted 386 cases as class 1 when they were, in fact, class 0 (false positives) and 129 instances as class 0 when they were, in fact, class 1 (false negatives). The comparatively low number of false positives (386) and false negatives 129 indicates that the model did well in error reduction.

The model's capacity to produce correct predictions for both class 0 and class 1 is further demonstrated by the balanced levels of true positives in both classes. Table 6 shows us a comparison between the search results using the bat algorithm in selecting features and without using it.

Table 7: Compare search results before selecting features and after selecting features

Models	Before features selection							
	Training				Testing			
	Accuracy	Recall	Precision	F1	Accuracy	Recall	Precision	F1
KNN	0.805	0.897	0.549	0.821	0.675	0.768	0.650	0.704
Adaboost	0.659	0.604	0.678	0.639	0.647	0.599	0.666	0.630
Decision Tree	0.998	1	0.996	0.998	0.838	0.940	0.782	0.854
Random Forest	0.998	1	0.996	0.998	0.903	0.941	0.876	0.908
Multi Layered Perceptron	0.630	0.549	0.654	0.597	0.622	0.551	0.646	0.595
Models	After features selection							
	Training				Testing			
	Accuracy	Recall	Precision	F1	Accuracy	Recall	Precision	F1
KNN	0.788	0.877	0.744	0.805	0.674	0.770	0.648	0.704
Adaboost	0.642	0.564	0.666	0.611	0.630	0.560	0.655	0.603
Decision Tree	0.997	0.999	0.994	0.997	0.852	0.953	0.795	0.867
Random Forest	0.979	0.986	0.972	0.979	0.859	0.930	0.816	0.869
Multi Layered Perceptron	0.617	0.567	0.628	0.956	0.605	0.567	0.617	0.591

IV. DISCUSSION

With regards to software quality measurement, Conventional strategies frequently depend on emotional evaluations, consume critical time and assets, have a restricted degree, and battle to deal with the intricacy of current programming frameworks. These limits make an examination hole that should be tended to by proposing another strategy that can give goal and far reaching estimations of programming quality. The investigation of alternative strategies to increase the automation, efficiency, and accuracy of software quality measurement, such as the bat optimization algorithm, is where there is a research gap. The proposed method aims to fill this research gap, advance software engineering practices, and provide software development organizations with useful insights. The results indicate that achieved Decision tree 99.7% and Random forest classifier 97.9% training accuracy when we use bat algorithm to select best features in 10 iterations. While Ibrahim and others achieved the Grasshopper algorithm and the particle swarm algorithm in 500 iterations 99.2% and 98% respectively. The high results of the decision tree and random forest may be due to the fact that the data set we are dealing with in the part(class) Consists of two types of data (yes and no) and because the decision tree and random forest in their work, in every step, we face two options without a third, which enabled them to reach the best Solutions. Table 7 shows us a comparison between the search results using the bat algorithm in selecting features and without using it. These models use the ensemble approach to increase accuracy and resilience by integrating numerous decision trees. KNN, on the other hand, employs instance-based learning, in which categorization decisions are made based on similarities to neighboring instances. To increase performance, adaboost uses boosting, which focuses on misclassified cases. The poorer performance of Multilayered Perceptions suggests that more modification of its architecture and hyper-parameters may be necessary to get better outcomes. The research concludes by demonstrating the effectiveness of machine learning models utilizing bat algorithms, namely Decision

Trees and Random Forest, in assessing software quality. Software developers and quality assurance teams may benefit greatly from the correct identification and classification of software instances by using these models. Notwithstanding, some constraints exist and more investigation is necessary, such as the investigation of substitute classifiers and optimization methodologies to augment the precision and dependability of software quality evaluation

V. CONCLUSION

In this proposed, the software quality was evaluated using ML classifiers and the BAT optimization approach. The results shown that several classifiers had varying degrees of success in terms of accuracy, recall, precision, and F1 scores, Decision Tree and Random Forest shows outperformed the others. The significant part of the characteristics were removed by using the bat algorithm during the feature selection process, which developed in significant time savings. The training accuracy for the Random Forest classifier is 97.9% and the Decision tree is 99.7%. The key limitation of this proposed in the data set utilized for evaluation could have affected how well the classifiers performed. In the future work enhance the performance of the models, further optimization strategies like the Whale or Grey Wolf algorithms may be researched. These algorithms may provide different optimization and search strategies, which might improve software quality assessment.

VI. REFERENCES

[1] Fatih Gurcan and Cemal Köse, "Analysis of software engineering industry needs and trends", Implications for education International Journal of Engineering Education 33(4): 2017, 1361-1368.

[2] Giuliano C., Cristina C., Peter D. et al, "Current and Future Challenges of Software Engineering for Services and Applications ", Elsevier B.V., Procedia Computer Science 97 (2016) 34 – 42, CF2016, 18-20 October 2016, Madrid

[3] Rashad, A. F., & Umar, S. U. (2023). software quality measurement: A Comprehensive review. AS-Proceedings, 1(2), 216-231.

[4] Kumar Jakhar and K. Rajnish, "Software Fault Prediction with Data Mining Techniques by Using Feature Selection Based Models", International Journal on Electrical Engineering and Informatics - Volume 10, Number 3,2018.

[5] Ljubomir Lazic, Nikos E. Mastorakis, "Optimal SQM:Integrated and Optimized Software Quality Management" sweat transactions on information science and applications. 2015.

[6] Saleh Ibrahim Ahmed, Alsaif Omar Ibrahim, Thanoon Kifaa H.," Deep Coverage Strategy for Private Wireless Network Power Using Hybrid (Salp Optimization – Genetic) Algorithms" TRKU Volume 62, Issue 03, April, 2020

[7] X.-S. Yang, Nature-Inspired Metaheuristic Algorithms, Luniver Press, Middlesex University, UK, 2010

[8] R. V. Rao, V. J. Savsani, and D. Vakharia, "Teaching–learningbased optimization: an optimization method for continuous non-linear large scale problems," Information Sciences, vol. 183, no. 1, pp. 1–15, 2012.

[9] Mohammed, H. M., Umar, S. U., & Rashid, T. A. (2019). A systematic and meta-analysis survey of whale optimization algorithm. Computational intelligence and neuroscience, 2019.

[10] Yang X-S (2010) A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization (NICSO 2010). Springer, pp 65-74

[11] Umar, S. U., & Rashid, T. A. (2021). Critical analysis: bat algorithm-based investigation and application on several domains. World Journal of Engineering, 18(4), 606-620.

[12] Bestoun S. Ahmed, Kamal Z. Zamli and Chee Peng Lim, "Constructing A T-Way Interaction Test Suite Using the Particle Swarm Optimization Approach", International Journal of Innovative Computing, Information and Control, ISSN 1349-4198, Volume 8, Number 1(A), pp. 431-451. 2012.

[13] Romi Satria Wahono and Nanna Suryana, "Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction", International Journal of Software Engineering and Its Applications, Vol.7, No.5 (2013), pp.153-166

[14] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "Software measurement data reduction using ensemble techniques", Neurocomputing, vol. 92, (2012), pp. 124-132.

[15] C. Seiffert, T. M. Khoshgoftaar and J. Van Hulse, "Improving Software-Quality Predictions With Data Sampling and Boosting", IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 39, no. 6, (2009), pp. 1283-1294.

[16] Deepti Arora and Anurag Singh Baghel, "Application of Genetic Algorithm and Particle Swarm Optimization in Software Testing", IOSR Journal of Computer Engineering (IOSR-JCE) eISSN: 2278-0661, p-ISSN: 2278-8727, Volume 17, Issue 1, Ver. II (Jan – Feb. 2015), PP 75-78
[17] R. Kumar Sahoo, D. Ojha, D. Prasad Mohapatra and M. Ranjan Patra, "AUTOMATED TEST CASE GENERATION AND OPTIMIZATION: A COMPARATIVE REVIEW", International Journal of Computer Science & Information Technology (IJCSIT) Vol 8, No 5, October 2016
[18] K. Senthil Kumar and Dr. A. Muthukumaravel, "Optimal Test Suite Selection using Improved Cuckoo Search Algorithm Based on Extensive Testing Constraints", International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 9 (2017) pp. 1920-1928
[19] Bestoun S. A., Luca M. G., Wasif A.; and Kamal Z. Z, "Handling Constraints in Combinatorial Interaction Testing in the presence of Multi Objective Particle Swarm and Multithreading", Information and Software Technology Journal, arXiv:1804.07693v1 [cs.SE] 20 Apr 2018
[20] Mehdi Gheisari, Deepak Panwar, Pradeep Tomar, Harshwardhan Harsh, Xiaobo Zhang, Arun Solanki, Anand Nayyar, Jafar A. Alzubi, "An Optimization Model for Software Quality Prediction with-Case Study Analysis Using MATLAB" DOI 10.1109 /ACCESS .2019. 2920879, IEEE.
[21] Saleh, I. A., H AL_Bayati, A., & Hadi Thanoon, K. (2020). Measure the software quality based on grasshopper optimization algorithm. International Journal of Computing and Digital Systems, 10, 2-8.
[22] Saleh, I. A., & Mahammead, S. R. (2018). Apply Particle Swarm Optimization Algorithm to Measure the Software Quality. AL-Rafidain Journal of Computer Sciences and Mathematics, 12(1), 26-36.
[23] Yang X-S (2010) A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization (NICSO 2010). Springer, pp 65-74
[24] Xin-She Yang, Nature-Inspired Optimization Algorithms, 2014, Pages 141-154.
[25] B. Mahesh, "Machine learning algorithms-a review," International Journal of Science and Research (IJSR)[Internet], vol. 9, pp. 381–386, 2020
[26] T. O. Ayodele, "Types of machine learning algorithms," New advances in machine learning, vol. 3, pp. 19–48, 2010.
[27] S. Ray, "A quick review of machine learning algorithms," in 2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon), IEEE, 2019, pp. 35–39.
[28] J. Zhang and J. Cheng, "Study of Employment Salary Forecast using KNN Algorithm," in 2019 International Conference on Modeling, Simulation and Big Data Analysis (MSBDA 2019), Atlantis Press, 2019, pp. 166–170.