Volume 13, No.5, May, 2026

Journal of Global Research in Mathematical Archives

ISSN 2320 – 5822

RESEARCH PAPER

Available online at http://www.jgrma.com

REIMAGINING MAINTAINABILITY MACHINE LEARNING TECHNIQUES FOR SECURITY REQUIREMENT OPTIMIZATION

Gopal Verma
MTech. Scholar
Millennium Institute of Technology and Science
Bhopal, Madhya Pradesh, India
gpl.vrm123@gmail.com

Atul Kumar Mishra
Assistant Professor
Millennium Institute of Technology and Science
Bhopal, Madhya Pradesh

Abstract: Software maintainability has gained recent popularity in the sphere of software engineering throughout the past few years in an effort to determine the quality of software. Hence, it is important to predict this maintainability in time and with accuracy for the effective administration of software during the maintenance stage. In turn, it is causing the developer to focus more on those modules that are expensive to maintain. software maintainability prediction (SMP) machine learning model suggested in this paper is informed by the Students project requirements software requirements dataset. This study is a description of the use of machine learning high-end methods of classification namely the Random Forest, AdaBoost and Voting Classifier which significantly contribute to the evaluation of maintainability in terms of software security requirements. To make a comparative analysis, these models have significant accuracy improvement with the highest accuracy of 87.85 in binary classification and 99.37 in multi-class classification compared to 79.43 and 85.08 of the baseline models respectively. The results show that more ML methods can be used to enhance the efforts to measure the maintainability of the software and that these methods can be used to meet the requirements of software security.

Keywords: Soft Software Maintainability, Metrics, security, Software, machine learning.

1 INTRODUCTION

Software maintenance leads to increased costs to developers in the software development life cycle since more than 70 per cent of the time is devoted to software maintenance [1]. The software has however been growing in size and complexity which makes it harder to update and seal security gaps [2][3]. The maintainability of the software of a product or business has a negative relationship with the success and profitability of the enterprise [4]. Maintainability may be used to evaluate software which is to be changed, or bugs are likely to be added after implementation. Transformational software maintainability is hence a quality attribute that adds to the overall performance of the software [5]. During maintenance, the developers have the opportunity to update workable estimates of the quantity of code that will change [6]. These forecasts allow software designs to foresee and make adjustments to the required adjustments in the future development of the modules.

Software maintainability is one of the issues that the computer industry is facing. This is the reason why so as to automate the system. Several ML and AI methods are still in application today [7][8]. The majority of the methods have used a limited number of software in conducting the tests [9]. Previous studies have shown that not a single study has been performed on the application of deep learning to predict software maintainability measures. It is also the predictability of the software maintenance that only makes minor improvements because of such operations [10]. Actually, the data is insufficient to draw certain conclusive results to enlighten on the system input data.

The respective research on Software Maintainability Metrics Prediction is primarily focused on the application of various ML algorithms, such as ANN and ridge regression, to two or three software models and compares the various software measures and machine learning algorithms [11][12].

The past has identified that there is no research on the use of DL to predict the software maintainability indicators. Based on the summary of research publication it has also been pointed out that there has not been much development in predictability of software maintenance.

1.1 Contribution of study

Among the contributions of this manuscript is the development of the maintainability of the software with the help of machine learning methods that are security-oriented and based on classification. The main contributions are:

- The software requirements in the context of security are met by using the sophisticated feature extraction methods which include tokenization, stemming and TF-IDF techniques.
- SMOTE technique is applied to the data to have effective classification of security requirements.

- Compares and contrasts the work of the various models such as RF, AdaBoost and Voting Classifier on the basis of F1-score, recall, accuracy, and precision as the evaluation measures.

1.2 Structure of Paper

In this study, the following sections have been structured in the following way: Section 2 provides an overview of the previous research on the topic of software maintainability in relation to the different approaches. The research methodology is indicated in Section 3 of this research. Present in Section 4 are the findings of the experiment, as well as the appraisals of the study project. Further on in the fifth version, findings and recommendations.

2 LITERATURE REVIEW

The literature background of the software maintainability was on a security requirement that centred on machine learning methods and techniques.

According to Sukkasem and Soomlek (2023), machine learning is one of the methods of evaluating the quality and maintainability of the software and the first results are encouraging. In this article, enhanced code smell design with the use of ML is developed. The results of the test reveal that the optimized ml classifiers achieve an accuracy of 99.18 (class-level) and 99.15 (function-level) accuracy in the classification and functional-level code smells respectively. During the process of determining data classes, the machine learning- based enhanced code smell classifier achieved a recall of 9.514, and in long approach, 98.806. The comparative findings further revealed that at least, as concerns the code smell identification, the improved machine learning classifiers were better than the original ones [13].

Silva, Bezerra, and Machado (2023) are concerned with the training of white boxes of the ML models that are aimed at predicting FM maintainability and provide 15 metrics that should be considered in this case. These are the steps that they are using to build the models. In the first place, they developed a human oracle on FM maintainability classifications to determine two procedures to evaluate FM maintainability. Thereafter they optimally classified the training data of the MLs, trained and evaluated 3 MLs, and compared the measures of recall, precision, F1, classification accuracy, and AUC-ROC. They took the optimal model in order to build it into a feedback mechanism for the domain engineers who indicated areas of improvement. The authors found the best model using the decision tree method and identified precision and accuracy as well as recall of 0.81, F1-Score of 0.79, and AUC-ROC of 0.91 to be the most important values of the model. FM restructurings is also an important system to develop so as to ensure artefact maintenance [14].

In the current paper, Gupta et al. (2021) pay attention to feature sets contained in the code and created models to forecast the eight categories of code smells. They also depict the application of feature selection to construct a more practical feature set and methods of addressing the issue of class imbalance by sampling the data [15].

Past studies applied such algorithms as Random Forest and Naive Bayes, yet they did not explore the deep learning approaches to predicting code smell. This paper produced the SMOTE data and discovered that the deep learning models not only obtained high level of accuracy and AUC, but also experienced an increase in accuracy within the range of 88.47 to 96.84.

The authors in Gupta and Chug (2021) compare the performance of five various BAs (XGB, AdaBoost, LightGBM and CatBoost) with that of SMP by using publicly available datasets. The RMSE, MMRE, and Pred(0.25), Pred (0.30) and Pred(0.75) measures were used to determine the prediction accuracy. Besides, the authors compared performance prediction of the proposed models. Although GBM and LightGBM were the best in terms of the RMSE, XGB was the best in 6 of the 7 datasets (85.71 of the total) because its MMRE values were between 0.90 and 3.82, and it was the lowest in residual errors in the datasets. Moreover, in the case under consideration, various BAs proved to be very different. XGB and CatBoost, in this instance, were evidently the winners as depicted by the post hoc analysis conducted through the use of pertinent statistical tools [16].

Baker et al. (2019) apply tools of ML in this work to develop a functional classification method of software requirements. Precisely, they discuss the research and development and the working implementation of two neural network models, CNN, and ANN, to categorise the NFRs as 5 categories, namely, operability, performance, security, usability and maintainability.

They show their most desired datasets containing approximately 1000 NFRs, as well as test their methodology on this. The CNN model applied in this case had a success rate of 82% in NFRs classification that was tested again with an F-score, recall and accuracy of 92, 76-97 and 82-94 respectively [17].

TABLE I. RELATIVE LITERATURE ON SOFTWARE MAINTAINABILITY IN TERMS OF MACHINE AND DEEP LEARNING TECHNIQUES.

Author(s) & Year	Dataset	Method	Performance	Limitations / Remarks
Sukkasem & Soomlek (2023)	74 open-source projects	Decision Trees, Random Forests with Particle Swarm and Bayesian Optimization	Accuracy: 99.18% (class-level), 99.15% (function-level); Recall: 95.14% (data class), 98.81% (long method)	Handles only a few types of code smells; limited optimization scope; more optimization methods could be explored.

Sukkasem & Soomlek (2023)	FM maintainability dataset, human-based oracle	Decision Trees	Precision: 0.81; Accuracy: 0.81; Recall: 0.81; F1: 0.79; AUC-ROC: 0.91	Restricted to FM maintainability; future research could extend method to other software types and improve feature selection.
Gupta et al. (2021)	Code smell datasets	Deep Learning, SMOTE	Accuracy: 88.47%-96.84%; AUC improved with deep learning models	No hyperparameter optimization performed; future work may include advanced deep learning models and tuning.
Gupta & Chug (2021)	7 open-source datasets	Boosting Algorithms (XGB, AdaBoost, LightGBM, CatBoost)	XGB performed best in MMRE for 6/7 datasets; GBM & LightGBM had lowest RMSE	Small datasets; future studies could use larger, more diverse datasets and additional performance metrics.
Baker et al. (2019)	2 popular NFR datasets (~1000 NFRs)	CNN, ANN	CNN: F-score 82%-92%; Recall 76%-97%; Accuracy 82%-94%	Limited to 2 datasets; performance may vary with different data or architectures.

Gap Analysis: Table 1 The Summary of Background Research on Software Maintainability. Up-to-date application of ML and DL has demonstrated the continued gaps in research level in software maintainability and code smell detection. The existing models are incapable of working with complex systems since most of them are trained on small datasets and merely obsess with code smells. Even though some improvements have been achieved, such as hyperparameter tuning, and other such techniques, the absence of sophisticated deep learning models and real-time applications is also not discussed. Inequalities in society and fast-changing environments will be researched and planned. In order to harness the complete potential that ML and DL have on software engineering, studies will need to implement more sophisticated deep learning methods, combine systems to run continuous maintainability, and expand the data sets.

3 METHODOLOGY

To apply machine-learning methods of classifying security requirements and enhance the maintainability of the software (the target of the studies and research), there are a number of steps and phases to be observed. To implement a research design, one must first collect data on a set of student software project which include security and non-security requirements. This will be followed by a preprocessing of the data, i.e., tokenization, stop words, stemming and TF-IDF with respect to converting the data to meaningful features. There is then a process of balancing the dataset with SMOTE and then encoding categorical variables with a label encoder to numeric coding. The data is divided into training and test data 80 and 20 percent respectively. Thereafter, machine learning (ML) models are applied, which include RF, Adaboost, and voting classifier, and the models are tested under the performance metrics of recall, f1-score, accuracy, and precision. The whole methodology is illustrated in Fig 1.

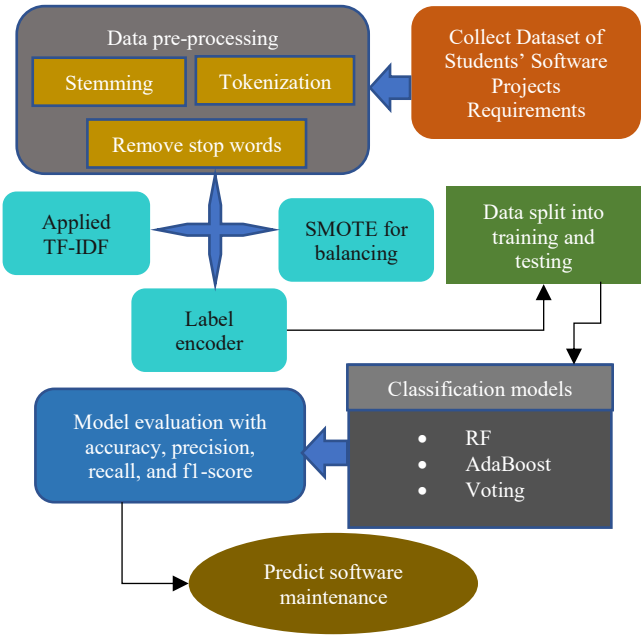


Figure: 1 Proposes the Flowchart that Enlists the Implementation of the Entire Research.

The overall research design procedure of software maintainability is as follows:

1.3 Data Collection

Software Projects Data Software Projects Requirements. The dataset has 1317 software requirements, 801 NSR and 516 SR, which are approximately 61 and 39 per cent of the dataset, respectively. The data consists of two versions of the original data, one of which is binary-classified (NSR and SR, too) and the other is multiclass-classified.

1.4 Data Preprocessing

Data processing aims at adding value and informative content to the raw data through the enhancement of its basic properties. The following is the list of the many preprocessing operations that this study employed to clean the data to be analyzed and modelled:

Tokenization: Tokenization makes the understanding of a single word in a sentence at a time by a machine rather simple than understanding a complete corpus by subdividing it into smaller bits.

Removal of Stop words: Stop words were eliminated such as the, is, are, etc. A stop word is a word that has been over-utilised within a text, yet, it actually means nothing and does not add any meaning to the text.

Stemming: This is the step of dimensional reduction of feature sets by substituting similar words with roots. The process decreases the dimensions of the feature sets.

1.4.1 Applied TF-IDF

Term Frequency tf(t,d): This is also known as tf and it is the number of times a term is repeated within a document. This is also known as frequency of document d. It is derived in Equation (1).

$$TF\ (t,d) = \frac{count\ of\ t\ in\ d}{number\ of\ words\ in\ d} \qquad (1)$$

Inverse Document Frequency idf(t,D): In document (collections) the frequent words or terms receive five weights and infrequent receive more weights. This can also be explained in Equation (2). The number of times a given term is used in a document is defined as the inverse logarithmic percentage of the overall documents in which the given term is present in.

$$idf\ (t,D) = log\ \frac{N}{df+1} \qquad (2)$$

Equation (3) is of Term Frequency - Inverse Document Frequency: The product of the frequency of the words in the document and the inverse document frequency. ML algorithms produce bi-gram transcripts and uni-gram transcripts with the help of the TF-IDF vectorizer.

$$TF - IDF = tf\ (t,d) * idf\ (t,D) \qquad (3)$$

1.4.2 Balance was done with SMOTE.

In case of balancing a small-sized data, it is effective to rely on the SMOTE. Nonetheless, when the size of the datasets is greater, it might not be prudent to use SMOTE because this might be more time-consuming in the process of generating the new samples. In addition, there is a high probability that SMOTE may generate overlapping dummy samples and, thus, become a redundant solution to the problem.it is mathematically represented in Equation (4).

$$x_{new,attr} = x_{i,attr} + rand(o,1)X\left(x_{iJ,attr} - x_i, attr\right) \quad (4)$$

1.4.3 Label Encoder

The more effective method of automating the process of converting categorical variables into numerical variables is to make a systematic definition of the values of each input category and give a numerical value when using the Label Encoder. Then there is the Label Encoder that assists in converting the raw categorical data to a more useful numerical format. A common methodology for this, is applied to the input features, target variables and class labels in a classification problem.

1.4.4 Train-Test Split

Once, there has been data preprocessing, you will have 2 sets of data, one for training the model, and another testing the model, 80% of the data is going to be used in the training set, to train the model, and 20 percent in the testing set, you can test the model.

1.4.5 Classification Models

To achieve software maintainability, a number of various strategies were outlined in the Machine Learning Classification Techniques for Security Requirements. These include:

1.4.6 Random Forest (RF)

Random Forest is one of the methods to solve the problems of classification and regression with the help of the supervised machine learning [18][19]. It also comes up with multiple decision trees during the training stage and employs the majority voting system to arrive at the decision, which subsequently enhances the accuracy of prediction and provides more uniform outcomes. We further

improve the value of Precision due to the application of Bootstrap aggregation with entropy criterion. In the case of random forest, the equations below can be calculated in Equations (5 and 6):

$$IG(N_p, a) = Gini\ (N_p) - \sum_{i-1}^{c} \frac{|N_i|}{|Np|} Gini\ (N_i) \qquad (5)$$

$$Gini(N_P) = 1 - \sum_{j-1}^{m} p_j^2 \qquad (6)$$

The data amount at node Np is denoted by Np, whereas the data quantity at node Ni is indicated by |Ni|. The number of distinct labels for the data at node Np is shown by the range $0 \leq I \leq c$, and the ratio of the number of data with the jth label to the total amount of data at node Np is represented by Pj. The number of the lab is indicated by the "j."

1.4.7 Ad Boost Classifier

AdaBoost as an ensemble learning method, can be applied to boost weak models both in regression and classification. It gives particular attention to wrongly classified training samples and less emphasis to the ones that were correctly classified. The forecasts of weak models are modified after aggregation, according to their accuracy. The algorithm is also quite efficient when it comes to unbalanced data and optimizing the performance of underperforming models. It is strong and flexible in numerous situations, and it usually leads to valid conclusions.

1.4.8 Voting Classifier

A Voting Classifier is a classifier that is trained to utilize various models and consequently predicts an output (class) that is probable to be an outcome (Figure 2). Each output class is located in all the classifiers sent to Voting Classifier and summed to get the resultant output class with the greatest number of votes. Rather than attempting to construct and refine two different models, with different accuracies, one model exists which is a combination of adaboost and decision tree methods and is used to generate an estimate by voting on the type of outputs. The other advantage of multiple classifier integration is the added accuracy and accuracy of predictions.

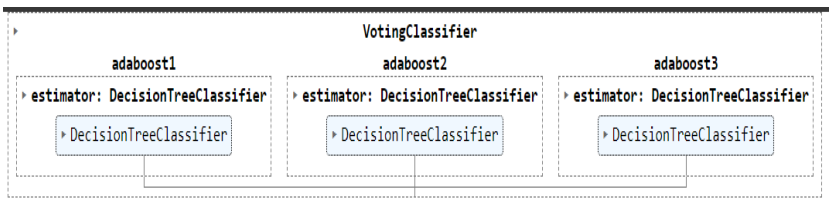


Figure: 2. Voting classifier

The second step is to incorporate information into a voting classifier.

1.4.9 Model Evaluation

Measures of evaluation give information regarding how well a model performs. We evaluated the proposed approach on the basis of a confusion matrix, accuracy, precision, recall, and the F1 score in this study. One way of summarizing the effectiveness of a classification system is to use a confusion matrix. Accuracy alone is fallacious when there is a strong imbalance between the occurrence of cases belonging to the various classes, or when this model contains more than two classes. One of the potential advantages of a confusion matrix is the capability to quickly determine whether a given system is prone to mislabel a particular class and the frequency, at most, of a pair or more classes being confused. The confusion matrix shown in Figure 3.

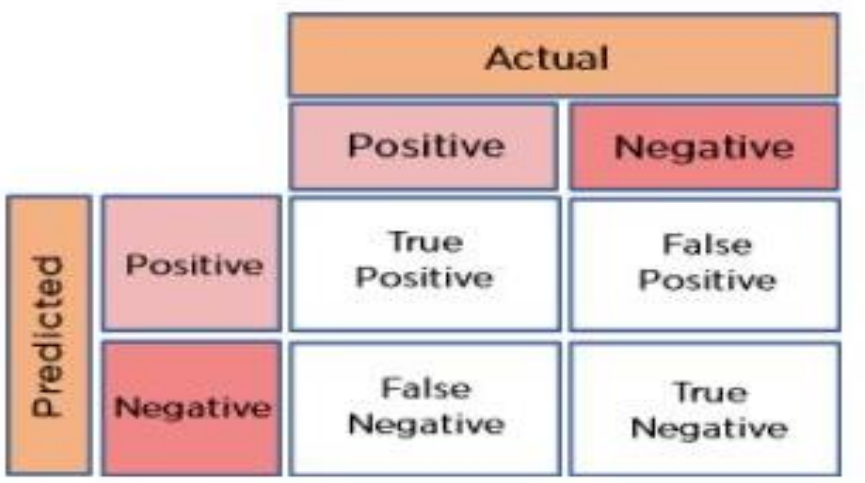


Figure: 3. Representation Class of Confusion Matrix

The system was successful in reaching the desired goals and outcomes in this study.

The comparison of the actual and expectation values will give the four different values, thus true negative (TN), false positive (FP), false negative (FN) and true positive (TP). One such fact is that given a scenario where an instance is supposed to possess diabetes but it is not the fact is perceived as a false positive. Accuracy, precision, and recall are calculated on the basis of values in the confusion matrix.

Accuracy: Accuracy indicates the proportion of accurate predictions based on the number of samples of the inputs, and it is Equation (7)-.

$$Accuracy = \frac{\text{TP+TN}}{\text{TP+Fp+TN+FN}} \quad (7)$$

Precision: Precision is described as the rate of the correct positive predictions that are attained relative to the total positive predictions attained. It is defined as Equation (8)-

$$Precision = \frac{\text{TP}}{\text{TP+FP}} \quad (8)$$

Recall: The sum of the relevant samples of all the correct positive results divided by the total correct positive results is referred to as recall. It is defined Equation (9)-

$$Recall = \frac{\text{TP}}{TP+FN} \quad (9)$$

F1 Score: F1 Score is used to establish the validity of a test which is defined as the Harmonic Mean of recall and precision and has a range of [0,1]. The strength and accuracy of your classifier is a measure of this. It is defined as Equation (10)-

$$F1 - Score = \frac{2(Precision*Recall)}{\text{Precision+Recall}} \quad (10)$$

The measures are used to define a quantitative document to compare the proposed ML model to the data.

4 EXPERIMENT AND DISCUSSION

My experiment setup was composed of an Intel Core i7 processor (3.4 GHz, 8 cores) and 16 GB RAM. The computer was configured to have Windows 10 and all the calculations were made in Jupyter Notebook with Python 3.9. Here, I provide the results as an analysis of the given experimental findings of the suggested machine learning models with the help of several graphs, figures, and tables.

1.5 Exploratory Data Analysis

EDA is the initial process in the development of any model. In the selection of machine learning algorithms, the identified hidden patterns in data will be useful in making better decisions. The best will be with the balance, feature scaling and correlation analysis in this instance.

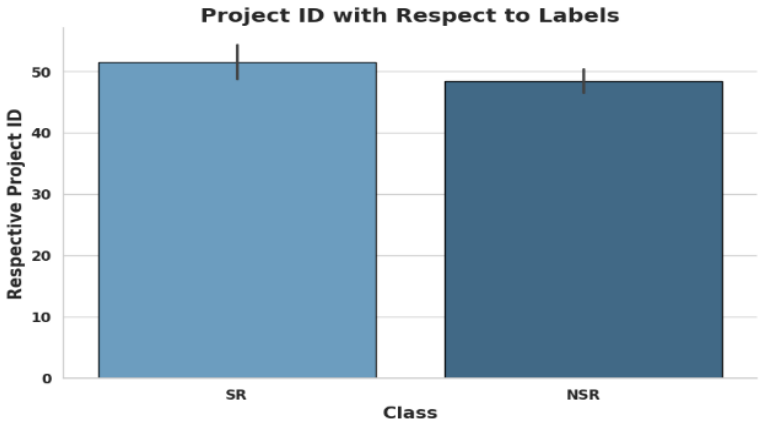


Figure: 4. Count Plot of Proetid Column

Fig 4 below presents the distribution of Project IDs against Labels. It provides the distribution of the Project IDs in two classes i.e. SR and NSR. The number of IDs in the 'SR' class is slightly above 50, whereas the number of IDs in the 'NSR' is slightly below 50. The error line showing the variability or confidence interval is also present in each of the bars.

The count plot of each label is presented in Fig 5 but the largest count is observed in the NSR label which approximately corresponds to 800. The following illustration is a frequency distribution of each label in the dataset, giving a clue as to the prevalence of each label. This could assist in determining whether additional checks on dataset balance are necessary.

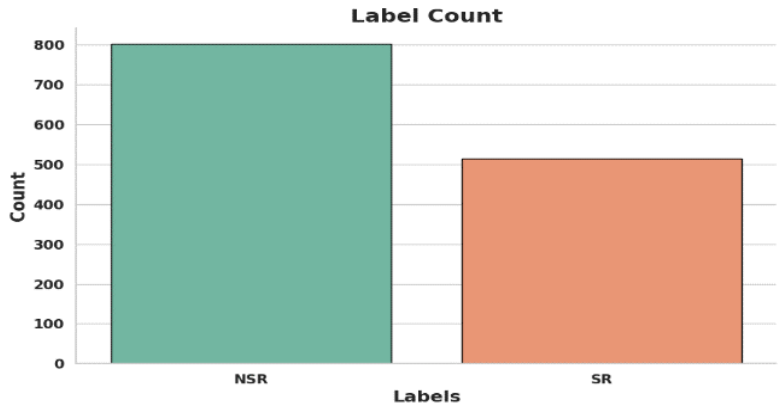


Figure: 5. Count Plot for the Labels

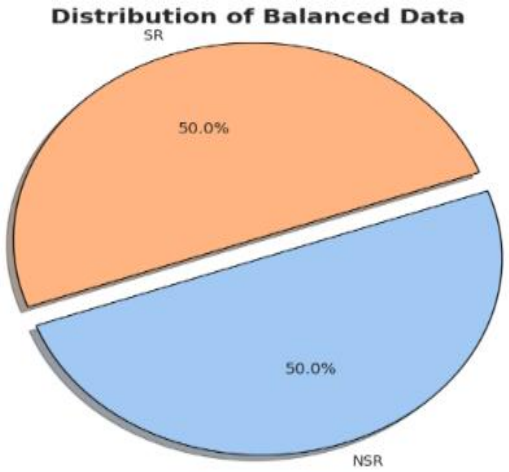


Figure: 6. Pie Chart for Data Distribution After Balancing

Fig 6 contains the pie chart that demonstrates post-data-balancing distributions. This information was skewed, and equal proportion distributions had been made which were utilized in this model. At this stage, both arbitrary groups, i.e. SR and NSR, have the same and complete data. Such equal representation is essential to make sure that the model is adequately trained and the bias is not presented in further machine learning applications.

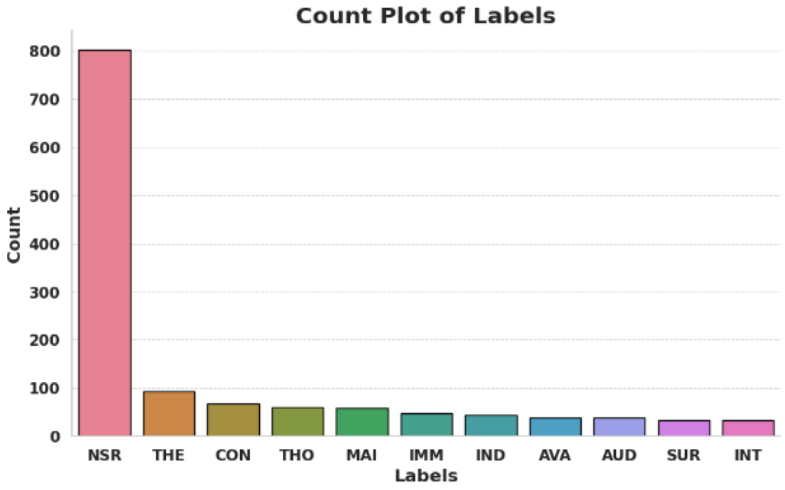


Figure: 7 Count Plot of Labels for Unbalanced Data

Fig 7 depicts the count plot of the balanced and unbalanced data in the dataset. The vertical axis is marked 0 to 800 and the horizontal axis indicates the labels NSR, THE, CON, THO, MAI, IMM and IND and AVA, AUD, SUR, INT and is marked with unbalanced data. The number of data under NSR label is much larger than the other counts. This tendency implies a lack of balance of data.

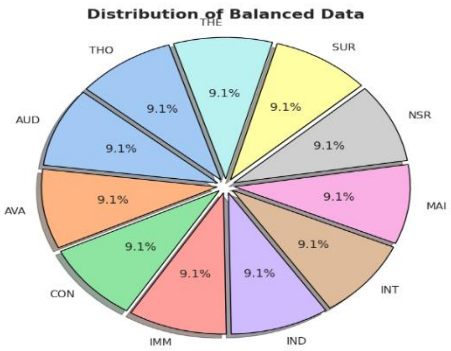


Figure: 8 Pie Charts for Distribution of the Data After Balancing

Fig 8 shows how the pie chart was allocated the balanced data. All the segments have equal shares which are 9.1 per cent. These are shortened into THO, SUR, NSR, MAI, INT, IND, IMM, CON, AVA or AUD. Equal distribution of data implies that the disproportion of the past data has been offset as in the equal distribution of data.

5 Results Data Analysis in The Binary Classes.

The following section provides the findings of binary classifier to the proposed ML classifier. According to Table 2 results, the voting classifier is the most accurate and it has an accuracy of 87.85 per cent when it comes to binary class as compared to 87.22 per cent in the case of the random forest classifier. The preciseness of the voting classifier is 88.32 per cent as compared to the random forest accuracy of 91.94 per cent. In this way, the voting classifier is more accurate when making predictions on positive instances. Voting classifier also has better recall of 87.85% when compared to the random forest so that the voting classifier has a larger F1 score of 87.83% and a better overall performance.

TABLE II. BINARY-CLASS CLASSIFICATION RESULTS OF ML MODELS

Measures	RF	Voting
Accuracy	87.212	87.865
Precision	91.924	88.372
Recall	82.523	87.859
F1-score	86.978	87.893

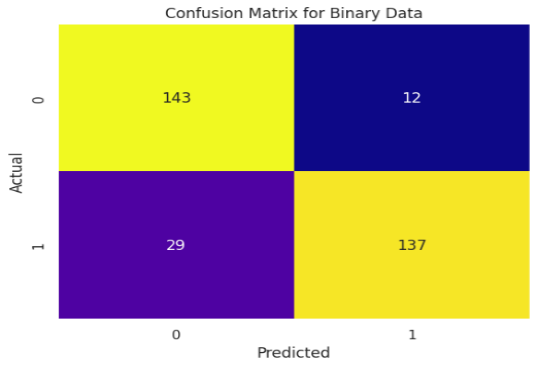


Figure: 9 Confusion Matrix Binary Class

In Fig 9, the performance of binary classification model is represented with 143 TN (true negatives), 137 TP (true positives), 12 FP (false positives), 29 FN (false negatives). It is advantageous to the model since it demonstrates the right and wrong guesses on the two categories enabling to assess the overall correctness.

Classification Report for Binary Data:

	precision	recall	f1-score	support
0	0.83	0.92	0.87	155
1	0.92	0.83	0.87	166
accuracy			0.87	321
macro avg	0.88	0.87	0.87	321
weighted avg	0.88	0.87	0.87	321

Figure: 10 Classification report for binary class

As observed on the binary class classification report in Fig 10, in Class 0, the precision, 0.92 recall, 0.87 F1-score and 155 support were achieved. In Class 1, the accuracy was 0.92, the recall 0.83 and the F1-score was 0.87 with the support of 166. In general, the model has worked well in both classes with the accuracy, the macro and weighted averages of the precision, recall and F1-score standing at 0.87, 0.88 and 0.87, respectively.

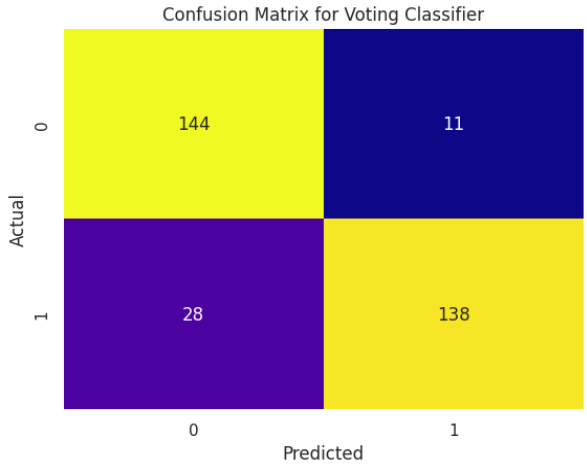


Figure: 11 Confusion matrix for voting on binary class

Based on the confusion matrix in Fig 11, the vote model had 144 TN (true negatives), 138 TP (true positives) and 11 FP (false positives), and 28 FN (false negatives). This assists in establishing the accuracy of the model by testing the right and wrong predictions made by each category.

Classification Report for Voting Classifier:

	precision	recall	f1-score	support
0	0.84	0.93	0.88	155
1	0.93	0.83	0.88	166
accuracy			0.88	321
macro avg	0.88	0.88	0.88	321
weighted avg	0.88	0.88	0.88	321

Figure: 12 Classification report for voting classifier on binary class

Figure 12 shows the voting classifier binary classification report given in section 2.1.3, the first class values are 0.84 and 155 support of precision, 0.93 of recall and 0.88 of F1 value. In the second class, which is precision, recall, F1 score, and support stand at 0.93, 0.83, and 166, respectively. The model appears to work fairly well with both classes with a general accuracy of 0.88, and overall support of 321.

1.6 Multi-class Result Analysis

The documentation will discuss the findings related to the suggested ML models for multi-class classification. Table 3 results show that the RF model is the most accurate one. The RF model performance measures have a close to ideal performance parameter with the model recording 99.37 F1 score and accuracy, precision, and recall of outstanding performance. By contrast, AdaBoost scores are much lower with the accuracy, precision, recall, and F1 score of all scores of 95.80, 95.78 and 95.78 respectively. In the case of the multi-class classification problem, RF model is verified to possess better performance metrics as compared to the AdaBoost model. Results of Table 3 ML model multi-class.

TABLE III. MULTI-CLASS CLASSIFICATION RESULTS OF ML MODELS

Measures	RF	AdaBoost
Accuracy	99.327	95.807
Precision	99.357	95.784
Recall	99.367	95.808
F1-score	99.379	95.786

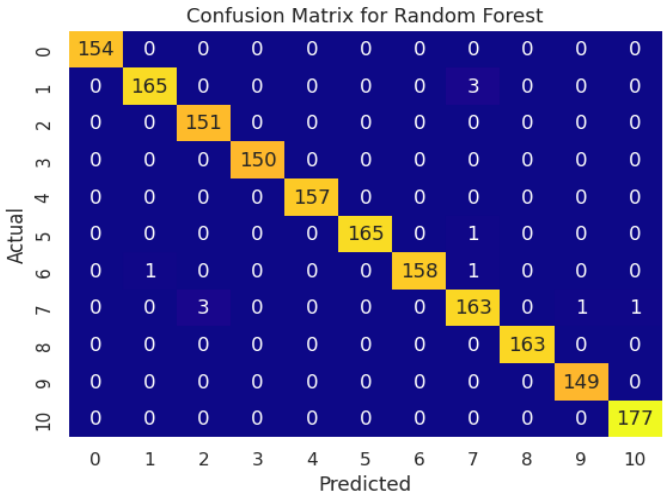


Figure: 13 Confusion matrix for RF on multi-class

The confusion matrix related to a multi-class classification task where an RF is used as a classifier is shown in Fig 13. It presents the predictive ability of the model and compares it with the real classes. The cells on the diagonal are correct instances that are to be predicted, and the off-diagonal cells are incorrectly classified ones.

Classification Report for Random Forest:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	154
1	0.99	0.98	0.99	168
2	0.98	1.00	0.99	151
3	1.00	1.00	1.00	150
4	1.00	1.00	1.00	157
5	1.00	0.99	1.00	166
6	1.00	0.99	0.99	160
7	0.97	0.97	0.97	168
8	1.00	1.00	1.00	163
9	0.99	1.00	1.00	149
10	0.99	1.00	1.00	177
accuracy			0.99	1763
macro avg	0.99	0.99	0.99	1763
weighted avg	0.99	0.99	0.99	1763

Figure: 14 Classification report for RF on multi-class

The classification report in Fig 14 indicates the performance of the classification problem in terms of most of the classes is of an exemplary nature, as the precision, recall and F1-scores of most of the classes were clustered between 0.97 to 1 and the report gives the model a score of perfection. All the classes registered a support of 149 to 166 which shows a good distribution of the data. The total accuracy of RF model was noted as 0.99 with a support of 1763.

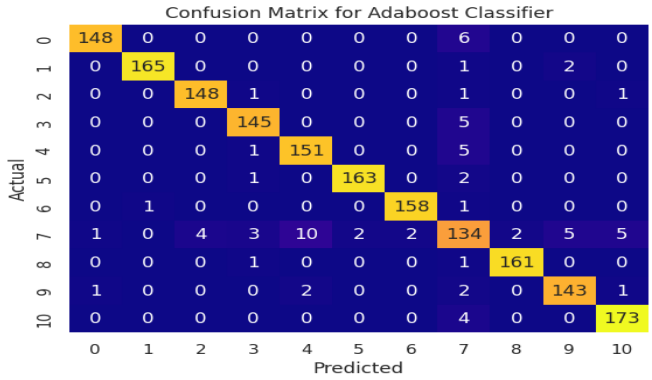
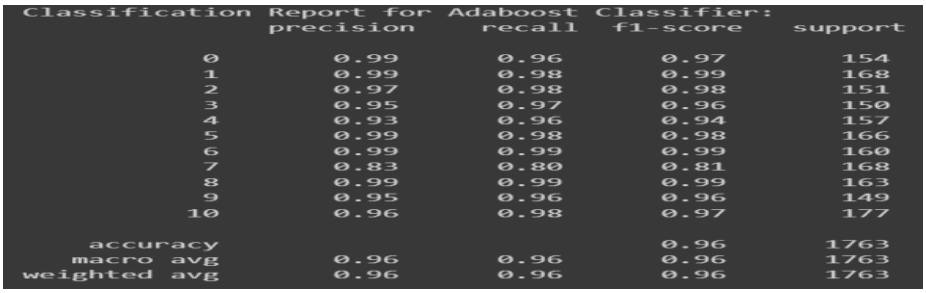


Figure: 15 Confusion matrix for AdaBoost on multi-class

The performance of the classifier for the various classes is depicted in Fig 15 which is the confusion matrix of a multi-class classification problem using an AdaBoost classifier. It indicates the cells that are the correct ones (diagonal in the figure), and are known as true positives, and the off-diagonal ones, which are known as false negatives.

Classification report of the AdaBoost model used in the multi-class classification problem (Fig 16) demonstrates excellent performance where most of the classes have a score of between 0.96-0.99 on precision, recall and F1-score which is close to perfection. Since all the classes are recorded between 154 and 177 support, it is an indication of a well-balanced dataset. The aggregate support of 1763 indicates that the ultimate accuracy of the AdaBoost model, 0.96 is sound.



Classification Report for Adaboost Classifier:

	precision	recall	f1-score	support
0	0.99	0.96	0.97	154
1	0.99	0.98	0.99	168
2	0.97	0.98	0.98	151
3	0.95	0.97	0.96	150
4	0.93	0.96	0.94	157
5	0.99	0.98	0.98	166
6	0.99	0.99	0.99	160
7	0.83	0.80	0.81	168
8	0.99	0.99	0.99	163
9	0.95	0.96	0.96	149
10	0.96	0.98	0.97	177
accuracy			0.96	1763
macro avg	0.96	0.96	0.96	1763
weighted avg	0.96	0.96	0.96	1763

Figure: 16 Classification report for AdaBoost on multi-class

1.7 Comparative study

This part will compare the software maintenance techniques based on the ML models for security requirements. The model performance with respect to binary or multi-class data is also taken into consideration.

TABLE IV. COMPARISON BETWEEN BINARY AND MULTI-CLASS CLASSIFICATION

Performance matrix	**Binary class**		**Multi-class**	
	RF	Voting	RF	AdaBoost
Accuracy	87.252	87.835	99.374	95.807
Precision	91.945	88.352	99.376	95.778
Recall	82.536	87.853	99.377	95.804
F1-score	86.984	87.836	99.374	95.783

The above Table IV illustrate results comparing binary and multi-class classification of the performance of the ML models. The models of the three classifications are RF, Voting, and AdaBoost, each of which has a variety of results. The voting model in binary classification has slightly better results of 87.85 than 87.22 and also a higher recall of 87.85 than 82.53. Nonetheless, the Random Forest model is more precise with the score of 91.94, as opposed to the Voting model that achieved the score of 88.32. In any case, the performance of Random Forest is far better than all AdaBoost criteria, having scored 99.37% in terms of precision, recall, accuracy, and F1-score, whereas AdaBoost only scores 95.80% in them. It means that Random Forest proves to be more effective when it comes to multi-class classification, and the Voting model has a slight advantage over binary classification.

TABLE V. COMPARATIVE ANALYSIS BETWEEN BASE AND PROPOSED MODELS FOR BINARY AND MULTICLASS CLASSIFICATION OF SOFTWARE MAINTENANCE

Binary classification				**Multi-class classification**			
Base models		Proposed Models		Base models		Proposed Models	
LR	NB	RF	Voting	LR	SVC	RF	AdaBoost
79.437	79.435	87.228	87.859	85.086	83.664	99.373	95.805

Table 5 shows both the results of base and proposed models in binary and multiclass classification of Software maintenance. In the suggested models of binary classifications, the accuracy of the linear regression and naive bayes models which gave the base settings of 79.43, increased significantly. The random forest and the voting classifier in the proposed models had an accuracy of 87.22 and 87.85 respectively. The linear regression and the support vector classifier that were the base models in the multiclass classification obtained accuracy of 85.08 and 83.66 respectively.

The suggested models, that is, Random Forest and AdaBoost proved to have significant high accuracy scores of 99.37% and 95.80% respectively, which demonstrates that there was a significant improved accuracy when the proposed models were used. It means that these models can be applied in binary and multiple classes classification problems.

6 CONCLUSION AND FUTURE SCOPE

Software development life cycle includes such a maintenance that is usually expensive and time consuming. When the software product is released to the client, software maintenance is started and ceases when the product is out of expected use. The maintainability of the software product is a quality attribute that determines the simplification of the process of incorporating changes within a given time frame. The major objective of the study is to prescribe using evolutionary algorithms, especially, ML algorithms, to forecast software maintainability and compare its performance with other ML algorithms. The study presented in this paper focuses on exploring the use of machine learning classification method to make the software more sustainable to meet the security requirements of the software. The base models versus the proposed models comparative analysis shows a considerable improvement in classification performance.

In comparison to the baseline models, the Random Forest, AdaBoost, and Voting Classifier models have significant increases in accuracy because AdaBoost and Voting Classifier achieve an accuracy of 87.85 and 99.37 percent in binary and multi-class classification respectively. Nonetheless, this research is small scale. The data sample is made of software projects done by students

and this is not necessarily a reflection of software systems in the real world in terms of their diversity and complexity. Still more persuasive evidence on the methods can be achieved and the methods can be improved by adding more other performance measures and by applying the models to the real-life situation.

REFERENCES

[1] R. Malhotra and A. Chug, "Software Maintainability: Systematic Literature Review and Current Trends," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 08, pp. 1221–1253, Oct. 2016, doi: 10.1142/S0218194016500431.

[2] Q. Huang, E. Shihab, X. Xia, D. Lo, and S. Li, "Identifying self-admitted technical debt in open source projects using text mining," *Empir. Softw. Eng.*, vol. 23, no. 1, pp. 418–451, Feb. 2018, doi: 10.1007/s10664-017-9522-4.

[3] V. Sanikal, "Scalable Cloud Based Infrastructure and Virtual ECU Driven Software Testing for the Next Generation Automotive Industry," *Int. J. Emerg. Res. Eng. Technol.*, pp. 88–92, 2025, doi: 10.63282/3050-922X.AECTIC-112.

[4] J. Guo *et al.*, "Data-efficient performance learning for configurable systems," *Empir. Softw. Eng.*, vol. 23, no. 3, pp. 1826–1867, 2018.

[5] A. Naresh, R. rao Thallada, and K. Nallabothu, "Risk-Based Governance for Autonomous Decision Systems," *J. Bus. Manag. Stud.*, vol. 8, no. 6, pp. 69–73, 2026, doi: 10.32996/jbms.2026.8.6.5.

[6] P. V. Bharati, J. S. V. S. Kumar, S. K. Anumula, P. V. Krishna, and S. Malla, "IoT and Predictive Maintenance in Industrial Engineering: A Data-Driven Approach," 2025. doi: 10.48550/arXiv.2511.04923.

[7] N. Kolli, J. W. Sajja, and A. Nerella, "Building Secure AI Agents for Autonomous Data Access in Compliance/Regulatory-Critical Environments," *Comput. Fraud Secur.*, vol. 2024, no. 9, pp. 363–373, Sep. 2024, doi: 10.52710/cfs.746.

[8] V. Methuku, S. Kamatala, P. Naayini, and P. R. Vontela, "From Ethical Principles to Technical Safeguards: A Unified Framework for Safe and Human-Centered Artificial Intelligence," *Am. Int. J. Comput. Sci. Technol.*, vol. 4, no. 5, pp. 26–34, Sep. 2022, doi: 10.63282/3117-5481/AIJCST-V4I5P103.

[9] V. K. Bollu, "Threat Landscape in Artificial Intelligence Systems: Taxonomy, Attack Vectors and Security Implications," *World J. Adv. Res. Rev.*, vol. 29, no. 1, pp. 285–294, 2026, doi: 10.30574/wjarr.2026.29.1.0007.

[10] A. A. Soni, M. Parikh, R. N. K. Dhenia, J. A. Soni, A. R. Jha, and S. M. Shah, "Reinforcement Learning for Dynamic Workflow Optimization in CI/CD Pipelines," in *2025 IEEE 17th International Conference on Computational Intelligence and Communication Networks (CICN)*, IEEE, Dec. 2025, pp. 638–644. doi: 10.1109/CICN67655.2025.11367872.

[11] M. Kari, "Deep Learning-Based Fault Prediction Models for Enhanced Network Security Monitoring," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 3, no. 3, p. 492, Jun. 2023, doi: 10.48175/IJARSCT-11600I.

[12] T. P. Patel, A. K. Elengovan, V. Ranganathan, M. Parikh, and D. Kole, "Self-Healing AI Systems Using Multi-Agent Learning," in *2026 International Seminar on Intelligent Business and Edge-Computing Research (ISIBER)*, 2026, pp. 7–12. doi: 10.1109/ISIBER68248.2026.11470173.

[13] P. Sukkasem and C. Soomlek, "Enhanced Machine Learning-Based Code Smell Detection Through Hyper-Parameter Optimization," in *2023 20th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, IEEE, Jun. 2023, pp. 297–302. doi: 10.1109/JCSSE58229.2023.10202124.

[14] P. Silva, C. Bezerra, and I. Machado, "Automating Feature Model maintainability evaluation using machine learning techniques," *J. Syst. Softw.*, vol. 195, p. 111539, Jan. 2023, doi: 10.1016/j.jss.2022.111539.

[15] H. Gupta, T. G. Kulkarni, L. Kumar, L. B. M. Neti, and A. Krishna, "An empirical study on predictability of software code smell using deep learning models," in *International conference on advanced information networking and applications*, 2021, pp. 120–132.

[16] S. Gupta and A. Chug, "An Extensive Analysis of Machine Learning Based Boosting Algorithms for Software Maintainability Prediction.," *Int. J. Interact. Multimed. Artif. Intell.*, vol. 7, no. 2, pp. 89–109, Dec. 2021, doi: 10.9781/ijimai.2021.10.002.

[17] C. Baker, L. Deng, S. Chakraborty, and J. Dehlinger, "Automatic Multi-class Non-Functional Software Requirements Classification Using Neural Networks," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, IEEE, Jul. 2019, pp. 610–615. doi: 10.1109/COMPSAC.2019.10275.

[18] A. Sekulić, M. Kilibarda, G. B. M. Heuvelink, M. Nikolić, and B. Bajat, "Random Forest Spatial Interpolation," *Remote Sens.*, vol. 12, no. 10, p. 1687, May 2020, doi: 10.3390/rs12101687.

[19] V. Rohilla, D. S. Chakraborty, and D. R. Kumar, "Random Forest with Harmony Search Optimization for Location Based Advertising," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 9, pp. 1092–1097, Jul. 2019, doi: 10.35940/ijitee.I7761.078919.

