[1]Ranjeetsingh Suryawanshi

[2]Amol Kadam

# Enhancing Software Defect Prediction accuracy using Modified Entropy Calculation in Random Forest Algorithm

***Abstract: -*** Imagine you are trying to classify software defect for a large dataset. How will you choose the best algorithm to do that? For the above problem we have various algorithms like Random Forest, Support Vector Machine, Neural Networks, Naive Bayes, K-Nearest Neighbours, Decision Tree, Logistic Regression etc. One of the most used methods is Random Forest algorithm, which uses multiple Decision Trees to make predictions. However, this algorithm relies on a complex calculation called Entropy, which measures the uncertainty in the data. Entropy is a function that uses natural logarithm which may be time consuming calculation. Is there a better way to calculate entropy? In this research, we have explored a different way to calculate the natural logarithm using the Taylor series expression. It is a series consisting of sum of infinite terms that approximates any function by using its derivatives. We further modified the Random Forest algorithm by replacing the natural logarithm with the Taylor series expression in the Entropy formula. We tested our modified algorithm on dataset and compared its performance with the original Entropy formula. We found that our modification in the algorithm has improved the accuracy of the algorithm on software defect prediction.

***Keywords:*** Random forest; decision tree; classification; prediction; entropy; Taylor series;

## I. INTRODUCTION

Software defects are bugs, errors, or faults in software that lead to incorrect results. Due to many tasks during software development are carried out by humans, which may arise many software defect problems. Defects can occur during the phases of software development such as requirement analysis, design, coding, testing, deployment, and maintenance. software testing is important for software products to maintain software quality and reliability. Rapid growth of software applications requires more testing which is expensive and time consuming. When it comes to software defect detection, software defect prediction techniques are considerably more economical than software testing and evaluations. Machine learning analyses input data and predicts output values using a variety of techniques. Machine learning techniques learn and enhance performance and contributing in the automation of the labelling process.

Many researchers and academician contributions show that the probability of identifying software defect prediction models may be cost effective than the probability of identifying defect through the software inspections. Software Defect Prediction is a field of ongoing study where algorithmic and data-level approaches are used to discover software defects. The features of each technique and their applicability to programming defect predictions throughout programming development life cycle phases are explained by Mahesh Kumar et al. Expert opinions are the quickest and most straightforward method to obtain software inspection, however there is a significant degree of prediction uncertainty and prediction may be influenced by personal biases. However, when dealing with large amounts of data, Machine Learning-based models enhance their prediction accuracy by adjusting their parameter

[1,2]Bharati Vidyapeeth Deemed To Be University, College of Engineering, India

[1]ranjeetsinghsuryawanshi@gmail.com

[2]akkadam@bvucoep.edu.in

*Correspondence: ranjeetsinghsuryawanshi@gmail.com

values [1] .Model performance can be enhanced or the likelihood of incorrect model selection is minimized using ensemble learning. Bagging, Boosting, and Stacking are the three basic types of ensemble learning. Bagging is used to create statistical distributions and confidence intervals. Bias can be minimized by using boosting. Stacking determines the optimal way to integrate machine learning models [2].

## II. RELATED WORK

There are numerous data level and algorithm approaches that can be used for building prediction models, including data balancing, feature selection and machine learning algorithms. The literature provides descriptions of contributions made on defect prediction with empirical and conceptual approach by many researchers and academician. Table 1 shows some researchers contribution for software defect prediction.

**Table 1. Summary of related works**

| **Classifier used** | **Dataset used** | **Evaluation Measures** | **Future Work/Scope** |
|---|---|---|---|
| SVM [3] | CM1, PC1, JM1, PC3, KC1, EQ and JDT | Precision, Recall, F1-score, and Accuracy | Model can be evaluated on different datasets to ensure its performance. |
| NB, J48, RF, SVM, AdaBoost [4] | Eclipse, Columba, and Scarab | F1-measure and ROC | Feature selection can be tested with deep learning |
| AdaBoost, KNN, LR, NB and RF [5] | PROMISE repository | Accuracy, F-Measure, Specificity, G Mean | can be extended to predict cross-project defects. |
| KNN, NB, SVM [6] | PROMISE repository | ROC curve | Can be tested on large datasets. |
| SVM, NB, RF [7] | CM1, KC1, MC1, PC1, JM1, MW1, PC2, | MCC, ROC, PRC, F1-score | Model can be build using deep learning |

Feature selection and data balancing plays important role in enhancing performance of defect prediction model. Software feature becomes significant for the prediction if the inaccuracy grows due to altering feature values. Robust machine learning models are developed through complex interaction between software attributes and Shapley values can be applied to compute the degree of complexity between these attributes[8]. To improve the accuracy of software defect prediction, the best selected attributes are incorporated into the Random Forest classification process to provide more accurate results[9]. The author introduces a paradigm for predicting software defects that is based on heterogeneous attribute selection and nested stacking. To enhance the model's performance Nested-Stacking applies heterogeneous attribute selection approaches together with normalization to improve quality of data[10]. Gayatri et al identify appropriate attributes using a decision tree induction. The subset of attributes is made up of every characteristic identified in the Decision Tree Induction Rule. The classifier performs better when it learns this additional feature set using the same models[11]. Pham et al work using probabilistic categorization that identify the class value that maximizes the class's posterior probability for a given set of features[12]. Chennappan et al talk about data balancing which bias the result for the major number of classes if the data are imbalanced[13].

Defect prediction seems to be a major challenge for large amounts of data; hence, many machine learning methods have been used to build prediction model [21] [22].

Random Forest performs classification by using several decision trees that work together to form a decision forest. With comparatively high accuracy, it can handle huge datasets and multi-variable inputs; it excels at controlling imbalanced datasets. Multiple decision trees vote for their preferred outcomes, and random forest enhances the relationship between decision trees[14]. Using the Random Forests approach, a limited number of rows are selected at random from the total amount of data, and a collection of decision trees is constructed for every module to produce classification output[15]. Premalatha et al proposes a novel categorization and prediction technique for improving accuracy which is based on the Cost Random Forest algorithm, which minimizes the impact of faults in irrelevant software components[16].

A Bayesian network is a probabilistic visual model that reflects a joint distribution of probabilities across a set of independent random variables[17]. Uncertainty may be effectively handled by Bayesian networks. Using conditional probability, a Bayesian network can represent the relationship between information components and learn from uncertain data[14].The support Vector Machine is a linear classifier which performs binary classification using data on border points of the hyperplane. Using the kernel approach, which implicitly translates inputs into highly dimensional vector spaces, support vector machines can perform non-linear classification beyond linear classification[14].To increase the precision of classification, the support vector machine tries to select the optimal hyperplane with the biggest margins between data instances[18].

The KNN classifier is a slow method since it learns during the assessment step and maintains data samples throughout the learning step.[14] The KNN method determines the class of samples to be categorized based on the samples that are closest to each other.[15] Since K remains positive, a collection of objects with label information is used to select the neighbours. [16] To create the ideal classifier, the AdaBoost classifiers algorithm uses weak learner to train a group of classifiers. [17] To train a software defect prediction model, Yan Gao et al recommends application that combines the Backpropagation Neural Network method with Adaptive Boosting. AdaBoost avoids the overfitting issue and is an effective predictor for data that is unbalanced.[18]

Multiple fundamental classifiers were combined by Yun ZHANG et al to create an integrated prediction model called as Combined defect predictor. Author used two types voting technique to build integrated model. First Ave voting aggregates confidence ratings and second Max Voting generates the highest confidence values from many fundamental classifiers[19]. Marwa Assim et al categorized data using eight machine learning methods that used percentage split and k-fold cross-validation with WEKA tool[20].

Discussing about unsupervised machine learning Xin Dong et al has explained K-Means clustering in very simple way. After choosing K cluster centers at random, place the element in the cluster that is the most equivalent to it by comparing its resemblance to the other clusters. Determine the cluster center for each cluster by taking the average of all the items within that cluster[2].

Some researchers build software defect model using deep learning approach. Through multi-layer processing, Deep Learning converts original minimal feature representation into a high-level feature. Using basic models, it can perform complicated categorization jobs[14]. Models of neural networks are derived from biological neural networks, which are made up of synapses and neurons. Neurons are modelled as nodes in a graphical network, with synapses acting as weighted edges connecting the nodes[17]. Artificial neurons contain artificial neural networks, which may be used as a non-linear classifier in machine learning applications. The input neurons are interconnected, allowing them to operate simultaneously by communicating with one another through signals and calculating output using a non-linear function[21].

## III. MATERIALS AND METHODS

In this project we have implemented a modified version of Random Forest algorithm, which is a machine learning method that uses multiple decision trees to make predictions and classify the data. Entropy is a measure used to quantify the impurity or randomness in a set of examples. Entropy calculation by using natural logarithm is:

$$E = -\Sigma p(y_i)\, log_2(p(y_i)) \quad (1)$$

We have modified the algorithm by using Taylor series expression to approximate the natural logarithm used in the entropy formula to measure the quality of each node split in the various split in Decision Tress. Figure 1 shows detail workflow for our proposed framework for software defect prediction

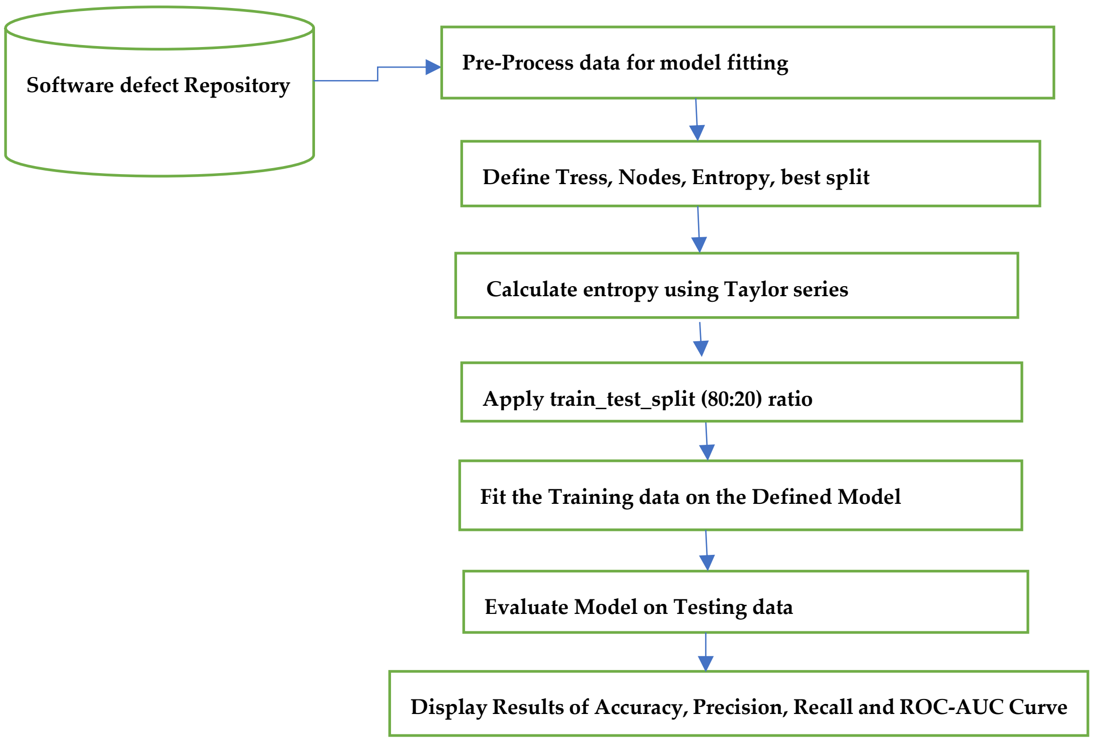

Software defect Repository
Pre-Process data for model fitting
Define Tress, Nodes, Entropy, best split
Calculate entropy using Taylor series
Apply train_test_split (80:20) ratio
Fit the Training data on the Defined Model
Evaluate Model on Testing data
Display Results of Accuracy, Precision, Recall and ROC-AUC Curve

**Figure 1. Proposed framework for software defect prediction**

### 3.1 Main Modules of Prediction model

**Class Node**: Define the structure for the node of decision tree

**Class Decision Tree**: Define the structure of decision tree and implement few functions for splitting, growing, and information gain from the dataset, etc.

**Class Random Forest**: Define the structure of random forest and implement few functions for fitting the model, predicting outcomes, etc.

**Main methodology**: For CM1 dataset, apply original and modified algorithm and evaluate and analyze the result and compare the performance of the modified algorithm with the original one.

**3.2 Data Source:**

We have used CM1 NASA dataset to test the performance of our model.

**NASA Software Defect Repository**: This is a publicly available dataset that was taken from a NASA-managed software defect repository. A wide range of NASA systems are included in the many NASA programmes. For example, CM1 stands for spacecraft instrumentation, whereas KC1, KC3, and MC2 stand for ground data storage management. The data transactions are managed by MW1, while the software for earth-orbiting satellites is denoted by PC1, PC2, PC3, and PC4. It includes a variety of attributes that are used as independent variables to determine whether a certain software is potentially flawed. These characteristics include, but are not restricted to, the program's complexity, the number of control flow statements, the number of lines of code, the number of comments contained within the code and many more attributes. The dependent variable 'Defective' represents the probability of a program being defective.

**3.3 Algorithm Selection and Modification:**

We chose the Random Forest algorithm as our base model because it has advantages such as reducing overfitting, handling missing values, and providing feature importance scores. We have modified the Random Forest algorithm by using the Taylor series expression to approximate the natural logarithm used in the Entropy formula. The Taylor series expression for natural logarithm is:

$$\text{Entropy} \;= \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}(x-1)^{n} \text{ from n = 0 to } \infty \qquad (2)$$

where, x is any positive number between 0 and 1. By using this approximation, we can calculate the natural logarithm for specific value of x using a finite number of terms.

The use of Taylor series gives much higher degree of accuracy in entropy calculation compared to the natural logarithm. On the other hand, the natural logarithm function requires a complex algorithm that involves iterative methods, floating-point arithmetic, and error handling. Moreover, the Taylor series allows to customize the approximation by choosing the degree of the polynomial. We have implemented this modification by creating a custom function that takes an array of probabilities as an input and returns an array of approximated logarithms as output. Then we have used the above function in our Entropy formula for calculation of each node split.

**3.4 Training and Testing Model:**

We have trained our modified Random Forest algorithm on training data using different hyperparameters such as number of trees (n_trees), maximum depth (max_depth), minimum samples split (min_samples_split). We then used a grid search technique to find the optimal combination of hyperparameters for each dataset. We have tested our modified Random Forest algorithm on CM1 training dataset and evaluated its performance using various metrics.

## IV. THEORY AND CALCULATION

We have selected 10 sample values from a CYCLOMATIC_DENSITY feature present in CM1 dataset. We will be finding the absolute value for each observation. The values are as follow:

[0.2, 0.13, 0.15, 0.17, 0.12, 0.2, 0.14, 0.28, 0.11, 0.17]. We can calculate entropy for value (0.2) as follow,

**E (0.2)** $= \frac{(-1)^{1+1}}{1}(0.2-1)^1 + \frac{(-1)^{2+1}}{2}(0.2-1)^2 + \frac{(-1)^{3+1}}{3}(0.2-1)^3 + \frac{(-1)^{4+1}}{4}(0.2-1)^4 + \frac{(-1)^{5+1}}{5}(0.2-1)^5 + \frac{(-1)^{6+1}}{6}(0.2-1)^6 + \frac{(-1)^{7+1}}{7}(0.2-1)^7 + \frac{(-1)^{8+1}}{8}(0.2-1)^8 + \frac{(-1)^{9+1}}{9}(0.2-1)^9 + \frac{(-1)^{10+1}}{10}(0.2-1)^{10} =$ **1.60.** Similarly, we can calculate for other points

**E (0.13)** $= \frac{(-1)^{1+1}}{1}(0.13-1)^1 + \ldots + \frac{(-1)^{10+1}}{10}(0.13-1)^{10} =$ **2.04,** **E (0.15)** $= \frac{(-1)^{1+1}}{1}(0.15-1)^1 + \ldots + \frac{(-1)^{10+1}}{10}(0.15-1)^{10} =$ **1.89**

**E (0.17)** $= \frac{(-1)^{1+1}}{1}(0.17-1)^1 + \ldots + \frac{(-1)^{10+1}}{10}(0.17-1)^{10} =$ **1.77,** **E (0.12)** $= \frac{(-1)^{1+1}}{1}(0.12-1)^1 + \ldots + \frac{(-1)^{10+1}}{10}(0.12-1)^{10} =$ **2.12**

**E (0.2)** $= \frac{(-1)^{1+1}}{1}(0.2-1)^1 + \ldots + \frac{(-1)^{10+1}}{10}(0.2-1)^{10} =$ **1.60,** **E (0.14)** $= \frac{(-1)^{1+1}}{1}(0.14-1)^1 + \ldots + \frac{(-1)^{10+1}}{10}(0.14-1)^{10} =$ **1.96**

**E (0.28)** $= \frac{(-1)^{1+1}}{1}(0.28-1)^1 + \ldots + \frac{(-1)^{10+1}}{10}(0.28-1)^{10} =$ **1.27,** **E (0.11)** $= \frac{(-1)^{1+1}}{1}(0.11-1)^1 + \ldots + \frac{(-1)^{10+1}}{10}(0.11-1)^{10} =$ **2.20**

**E (0.17)** $= \frac{(-1)^{1+1}}{1}(0.17-1)^1 + \ldots + \frac{(-1)^{10+1}}{10}(0.17-1)^{10} =$ **1.77**

**Table 2. Entropy calculation values by Taylor series**

| **Value** | 0.2 | 0.13 | 0.15 | 0.17 | 0.12 | 0.2 | 0.14 | 0.28 | 0.11 |
|---|---|---|---|---|---|---|---|---|---|
| **Entropy** | 1.6094 | 2.0402 | 1.8971 | 1.7719 | 2.1202 | 1.6094 | 1.9661 | 1.2729 | 2.2072 |

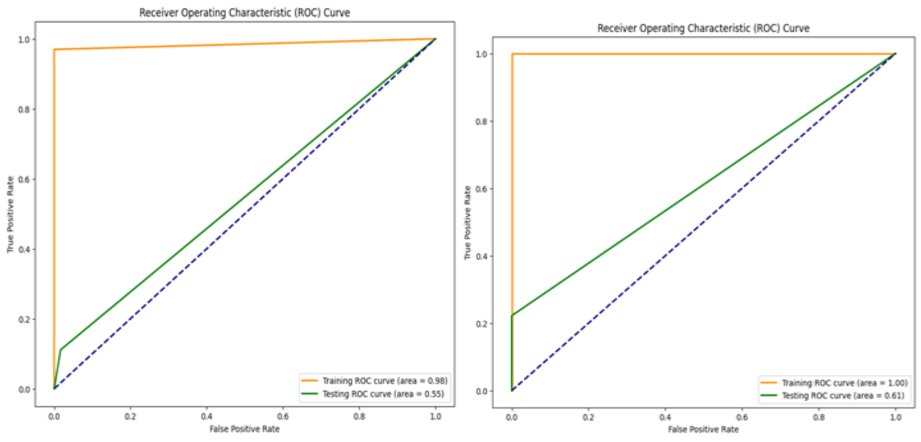
## V. RESULTS

The existing technique uses Entropy with natural logarithm to calculate the quality of each node split in the decision tree. Here we are using the modified Entropy using Taylor series formula because we want to explore a different way to calculate the natural logarithm used in the original Entropy formula. Table 3 shows result obtained on applying original Entropy formula and the modified Entropy formula using Taylor's series on CM1 dataset. From the results obtained by the random datasets, we can clearly observe the change in accuracy, precision and recall in the original Random Forest algorithm and the modified Random Forest algorithm.

**Table 3. Software defect prediction model result - Accuracy, precision, and recall**

| Parameters | Precision (0) | Precision (1) | Recall (0) | Recall (1) | Accuracy |
|---|---|---|---|---|---|
| Original formula | 0.88 | 0.5 | 0.98 | 0.11 | 0.8695 |
| Modified formula | 0.88 | 1 | 1 | 0.11 | 0.884 |

rom this table we can easily conclude that, we can improve accuracy with a calculation of Entropy using the Taylor series. **Figure shows** ROC curve for training as well as testing dataset for both original Random Forest Algorithm and modified Random Forest algorithm.



(a) (b)

**Figure 2. (a) ROC curve for original Random Forest Algorithm; (b) ROC curve for modified Random Forest Algorithm**

From the above ROC curves, we can conclude that area under the Training ROC curve and area under the Testing ROC curve has significant changes in their result.

## VI. CONCLUSIONS

This paper explored entropy calculation by using the Taylor series expression to approximate the natural logarithm used in the entropy formula to measure the quality of each node split. We have manually calculated entropy for selected 10 sample values from a CYCLOMATIC_DENSITY feature of CM1 dataset. We have implemented this entropy calculation in random forest algorithm and tested on CM1 dataset. Result indicate that it can improve accuracy and ROC curve shows that change in true positive rate which is good for testing dataset. Future work can be extended with predicting software defect in commercial dataset and building generalize model by testing software defect prediction on cross project.

**Author Contributions:**

**Ranjeetsingh Suryawanshi**: Data collection and analysis, developing methodology, Design and Development of an application. **Amol Kadam**: Reviewing and Editing of the article.

**Conflicts of Interest:** The authors declare no conflict of interest.

## REFERENCES

[1] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques," *Int. J. Appl. Sci. Eng.*, vol. 17, no. 4, pp. 331–344, 2020, doi: 10.6703/IJASE.202012_17(4).331.

[2] X. Dong, Y. Liang, S. Miyamoto, and S. Yamaguchi, "Ensemble learning based software defect prediction," *J. Eng. Res.*, no. November, 2023, doi: 10.1016/j.jer.2023.10.038.

[3] M. Mustaqeem and T. Siddiqui, "A hybrid software defects prediction model for imbalance datasets using machine learning techniques: (S-SVM model)," *J. Auton. Intell.*, vol. 6, no. 1, pp. 1–19, 2023, doi: 10.32629/jai.v6i1.559.

[4] S. K. Pandey and A. K. Tripathi, "An empirical study toward dealing with noise and class imbalance issues in software defect prediction," *Soft Comput.*, vol. 25, no. 21, pp. 13465–13492, 2021, doi: 10.1007/s00500-021-06096-3.

[5] K. K. Bejjanki, J. Gyani, and N. Gugulothu, "Class imbalance reduction (CIR): A novel approach to software defect prediction in the presence of class imbalance," *Symmetry (Basel).*, vol. 12, no. 3, 2020, doi: 10.3390/sym12030407.

[6] K. J. Eldho, "Impact of Unbalanced Classification on the Performance of Software Defect Prediction Models," *Indian J. Sci. Technol.*, vol. 15, no. 6, pp. 237–242, 2022, doi: 10.17485/ijst/v15i6.2193.

[7] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, “BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques,” *Expert Syst. Appl.*, vol. 144, p. 113085, 2020, doi: 10.1016/j.eswa.2019.113085.

[8] L. S. Shapley, “A Value for n-Person Games Contributions to the Theory of Games,” *In Annals of Mathematical Studies,edited by Harold William Kuhn and Albert William Tucker,Princeton University Press*, vol. 2, no. 4. pp. 307–318, 1953. doi: 10.1515/9781400881970-018.

[9] K. Magal.R and S. Gracia Jacob, “Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques,” *Int. J. Comput. Appl.*, vol. 117, no. 23, pp. 18–22, 2015, doi: 10.5120/20693-3582.

[10] L. qiong Chen, C. Wang, and S. long Song, “Software defect prediction based on nested-stacking and heterogeneous feature selection,” *Complex Intell. Syst.*, vol. 8, no. 4, pp. 3333–3348, 2022, doi: 10.1007/s40747-022-00676-y.

[11] N. Gayatri, S. Nickolas, and A. V Reddy, “Feature Selection Using Decision Tree Induction in Class level Metrics Dataset for Software Defect Predictions,” *World Congr. Eng. Comput. Sci. Vols 1 2*, vol. I, pp. 124–129, 2010, [Online]. Available: http://www.iaeng.org/publication/WCECS2010/WCECS2010_pp124-129.pdf

[12] D. T. Pham and G. A. Ruz, “Unsupervised training of Bayesian networks for data clustering,” *Proc. R. Soc. A Math. Phys. Eng. Sci.*, vol. 465, no. 2109, pp. 2927–2948, 2009, doi: 10.1098/rspa.2009.0065.

[13] R. Chennappan and Vidyaathulasiraman, “An automated software failure prediction technique using hybrid machine learning algorithms,” *J. Eng. Res.*, vol. 11, no. 1, p. 100002, 2023, doi: 10.1016/j.jer.2023.100002.

[14] H. Cao, “A Systematic Study for Learning-Based Software Defect Prediction,” *J. Phys. Conf. Ser.*, vol. 1487, no. 1, 2020, doi: 10.1088/1742-6596/1487/1/012017.

[15] T. D. Buskirk, “Surveying the Forests and Sampling the Trees: An overview of Classification and Regression Trees and Random Forests with applications in Survey Research,” *Surv. Pract.*, vol. 11, no. 1, pp. 1–13, 2018, doi: 10.29115/sp-2018-0003.

[16] H. M. Premalatha and C. V. Srikrishna, “Software fault prediction and classification using cost based random forest in spiral life cycle model,” *Int. J. Intell. Eng. Syst.*, vol. 11, no. 2, pp. 10–17, 2018, doi: 10.22266/IJIES2018.0430.02.

[17] L. Perreault, S. Berardinelli, C. Izurieta, and J. Sheppard, “Using classifiers for software defect detection,” *26th Int. Conf. Softw. Eng. Data Eng. SEDE 2017*, pp. 131–137, 2017.

[18] D. A. Pisner and D. M. Schnyer, *Support vector machine*. Elsevier Inc., 2019. doi: 10.1016/B978-0-12-815739-8.00006-7.

[19] Y. Zhang, D. Lo, X. Xia, and J. Sun, “Combined classifier for cross-project defect prediction: an extended empirical study,” *Front. Comput. Sci.*, vol. 12, no. 2, pp. 280–296, 2018, doi: 10.1007/s11704-017-6015-y.

[20] Q. O. and M. H. M. Assim, “Software Defects Prediction Using Machine Learning Algorithms,” *Int. Conf. Data Anal. Bus. Ind. W. Towar. a Sustain. Econ.*, pp. 1–6, 2020, doi: 10.1109/ICDABI51230.2020.9325677.

[21] S. P. Niculescu, “Artificial neural networks and genetic algorithms in QSAR,” *J. Mol. Struct. THEOCHEM*, vol. 622, no. 1–2, pp. 71–83, 2003, doi: 10.1016/S0166-1280(02)00619-X.

[22] Shinde, Swati V., and Deepak T. Mane. "Deep learning for COVID-19: COVID-19 Detection based on chest X-ray images by the fusion of deep learning and machine learning techniques." Understanding COVID-19: The Role of Computational Intelligence (2022): 471-500.