EVALUATION OF SOFTWARE BUG DETECTION USING RANDOM FOREST AND LOGISTIC REGRESSION CLASSIFIER

Nwachukwu-Nwokeafor Kenneth C[1]
Department of Computer Engineering,

Michael Okpara University of Agric, Umudike,
Nwachukwu.nkenneth @mouau.edu.ng, nwachukwuken72@gmail.com

Simeon Ozuomba[2]
Department of Computer Engineering,
University of Uyo, Akwa Ibom, Nigeria
simeonoz@yahoomail.com , simeonozuomba@uniuyo.edu.ng

Philip Asuquo[3]

Department of Computer Engineering,
University of Uyo, Akwa Ibom, Nigeria
philipasuquo@uniuyo.edu.ng

Stephen Bliss U.[4]

Department of Computer Engineering,
University of Uyo, Akwa Ibom, Nigeria
blissstephen@uniuyo.edu.ng

***Abstract*— The focus in this work is to provide efficient machine learning models that can be used to minimize or entirely eliminate bugs from software solution before they are deployed to the end users. Specifically, the Random Forest model (RFM) and the Logistic Regression Model (LRM) were considered in the study. The study used historical software metrics dataset obtained from the NASA Metrics Data Program (MDP) repository. The models were trained and evaluated for the baseline case without data balancing and also for the case were data balancing was implemented using Adaptive Synthetic Sampling (ADASYN). The results show that the RFM has higher prediction accuracy than the LRM. The prediction accuracies for the baseline were 73.68 % and 90.01% for the LRM and the RFM respectively. Equally, the prediction accuracies for the case with ADASYN-based data balancing were 85.05 % and 92.53% for the LRM and the RFM respectively. Hence, in respect of the prediction accuracy, the RFM with data balancing is recommended for the software bug detection in the case study dataset. However, the LRM has lower training time than the RFM in both cases. The training time of the RFM is about 18 times the training time of the LRM in the baseline case and about 31 times the training time of the LRM in the case with data balancing. In this wise, hybrid model that can take advantage of the low training time of the LRM and the high prediction accuracy of the RFM is recommended as topic for further studies.**

Keywords— Software Bug Detection, Random Forest Model, Logistic Regression Classifier, Adaptive Synthetic Sampling (ADASYN), Data balancing

1. INTRODUCTION

Over the years, software development processes and tools have improved with much focus on meeting the growing demand for high quality software solutions that are devoid of bugs [1,2]. Software bugs are such errors or flaws which can cause a software to deviate from it intended output or operation [3,4]. In some cases, software bug can cause minor damages but in some other cases the consequences of software bugs are so high resulting in life threatening situation, very high financial losses, and costly law suits [5,6]. In practice, software developers try as much as possible to detect and eliminate all possible bugs in a software solution before deployment [7,8]. However, it is always very difficult to achieve that by relying on manual checking of the software artifacts [9,10,11].

Consequently, over the years, several approaches and tools have been developed to facilitate efficient software bug detection [12]. One of such approaches in the present day is the use of machine and deep learning models for the detection or prediction of software bug [13,14]. The approach relies on the availability of historical dataset of the software artifacts. In addition, the performance of the models depends on a number of factors one of which is the highly imbalance nature of the software bug dataset. Accordingly, experts are relying on effective data balancing methods for the enhancement of the software prediction models [15,16]. Consequently, in this study, the Random Forest Model (RFM) and the Logistic Regression Model (LRM) are considered for software bug detection while the Adaptive Synthetic Sampling (ADASYN) is used for the data balancing [17,18,19]. The performance of the models in the baseline case without data balancing and in the case where the ADASYN-based data balancing are performed. In both cases, the training time is also considered to assess both prediction performance and the latency of the model. In all, the study use the key performance parameters to recommend the most suitable model for the software bug detection.

2. METHODOLOGY

The focus in this work is to use the Random Forest model (RFM) and then, the Logistic Regression Model (LRM) for detecting software bug based on the features present in a software bug dataset. The details of the dataset, the pre-processing and model training are presented in this section while section 3 presents the results of the model software bug predictions as well as the performance evaluation. The key components in this study are listed as follows:

(i) Data acquisition and description
(ii) Description of the software bug detection machine learning models: the Random Forest model (RFM) and the Logistic Regression Model (LRM)
(iii) Description of the data balancing approach: the Adaptive Synthetic Sampling (ADASYN)
(iv) Comparison of the performance of the two selected software bug detection models in the case of imbalanced data and in the case of balanced dataset

2.1 DATA ACQUISITION AND DESCRIPTION

The study used historical software metrics dataset obtained from the NASA Metrics Data Program (MDP) repository which is commonly used for predicting whether a software has defect or not. The target variable in the dataset is binary in nature, denoting whether a software module is fault-prone or non-fault-prone. The dataset is divided into various .arff files (example, CM1.arff, JM1.arff, KC1.arff, among others). The dataset is imbalanced and it also has so many features. Data balancing was done using the Adaptive Synthetic Sampling (ADASYN) method presented in section 2.3.

2.2 DEVELOPMENT OF THE MACHINE LEARNING MODELS FOR SOFTWARE BUG PREDICTION

2.2.1 Development of the Random Forest Model (RFM) for Software Bug Prediction

Random Forest Model (RFM) is an ensemble-based machine learning algorithm which combines the predictions of multiple decision trees to deliver robust and accurate results. In classification tasks such as imbalanced class problems, it is known for its resistance to overfitting and its ability to model complex decision boundaries. The decision tree in the RFM works by recursively partitioning the feature space using axis-aligned splits to minimize an impurity metric (Gini index in this research). Each split aims to improve class homogeneity. Given a dataset: $D = \left\{\left(x^{(i)}, y^{(i)}\right) \mid x^{(i)} \in R^d, y^{(i)} \in \{0,1\}\right\}, i = 1,2,3,\dots,n$, where $x^{(i)} = \left[x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \dots, x_d^{(i)}\right]$ is a feature vector and $y^{(i)}$ is the class label. To build a Decision Tree, at each internal node N, we seek the feature j and threshold t that minimizes an impurity criterion I, such as the Gini Index where:

$$Gini(S) = 1 - \left(\sum_{k=1}^{K} p_k^2\right) \quad (1)$$

Where, p_k is the proportion of sample in class k in subset S. Given the high variance of individual decision trees, RF introduces Bootstrap Aggregating (Bagging), where multiple trees are trained on bootstrap samples. Bootstrap sample $D_b \subseteq D$ is performed using sample with replacement for every bootstrap tree T_b and a decision tree is fitted on D_b with some randomization. The RF model prediction, $\hat{y}(x)$ is based on voting by majority as follows:

$$\hat{y}(x) = mode\left(\{h_b(x)\}_{b=1}^{B}\right) \quad (2)$$

Where, $h_b(x)$ is the prediction of the b^{th} tree and B is the number of trees. Variance in the model output is minimised by using bagging approach. The analytical expression for bagging is given as;

$$V_{avg} = \rho V_{indiv} + \frac{1-\rho}{B} V_{indiv} \quad (3)$$

Where, V_{avg} is the average variance, V_{indiv} is the individual variance, ρ is the average pairwise correlation between the base learners.

The strength of RFM lies in reducing ρ through feature randomization. At each split within a decision tree in RFM, only a random subset of features is considered. This further de-correlate the trees and avoids overfitting. For a node N, a subset of random features is selected as $\mathcal{F} \subset \{1, 2, \dots, d\}$. Among this subset, the best split is selected based on impurity. This randomness contributes to model diversity which is necessary in ensemble learning. Any $x^{(i)}$ that was not used in the bootstrap sample for tree T_b has its prediction included in the out-of-bag (OOB) estimate. It should be noted that OOB is used in this context to refer to the test set. Supposed B_i denotes a set of trees not trained on $x^{(i)}$, is given as:

$$\hat{y}_{OOB}^{(i)} = mode\left(\left\{h_b\left(x^{(i)}\right) \mid T_b \not\ni x^{(i)}\right\}\right) \quad (4)$$

Where, $\hat{y}_{OOB}^{(i)}$ is the OOB prediction output. This OOB error gives a reliable estimate of the model generalization without a separate validation set. RF by default tends to favour majority classes in imbalanced datasets. However, in this work, the Adaptive Synthetic Sampling (ADASYN) is used to increase the minority instances. Class weights are introduced to penalize misclassification of the minority class, while Gini index implicitly adjust when data distribution is balanced via oversampling. Consider a class imbalance expressed as:

$$Gini_{adj} = 1 - \sum_{k=1}^{K} \left(\frac{w_k n_k}{\sum_k w_k n_k}\right)^2 \quad (5)$$

Where, w_k is the weight of class k, and n_k is the number of samples in class k. If $\Delta x_j^{(b)}$ is used to describe the total decrease in impurity form feature j in tree T_b, then, feature importance scoring according to the Gini approach for RFM model is given as follows:

$$Importance(j) = \frac{1}{B}\sum_{b=1}^{B} \Delta x_j^{(b)} \quad (6)$$

In the use-case of this work, RFM attempts to identify which features are most decisive. The training time complexity analytical model for each tree is given as;

$$\tau_c = O(n \cdot d \cdot \log n) \quad (7)$$

The time complexity for training B trees with m features is given as:

$$\tau_c^B = O(B \cdot n \cdot m \cdot d \cdot \log n) \quad (8)$$

Where, n is the number of samples, d total number of features, and m is a subset of features in B trees.

2.2.2 Development of Logistic Regression Model (LRM) for Software Bug Prediction

Logistic Regression Model (LRM) is a binary classifier model. Unlike Linear Regression which predicts continuous outcomes, Logistic Regression predicts discrete outcomes by estimating the probability of class membership, constrained within the range [0,1]. In the context of this work, LRM is used to detect rare class instances, where the minority class is underrepresented. Hence, data resampling techniques like ADASYN is also explored to balance the data and identify relevant features. Let the dataset D be described as:

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m} \quad (9)$$

Where, $x^{(i)} \in R^n$ is the input vector for the i^{th} instance, $y^{(i)} \in \{0,1\}$ is the class label, and m is the number of training samples. The LRM adopts the response variable log-odds which is expressed in respect of the inputs as follows:

$$\log\left(\frac{P(y=1 \mid x)}{1-P(y=1 \mid x)}\right) = (\theta^T)(x) \quad (10)$$

Where, $\theta \in R^n$ is the parameter vector to be learned, and $P(\cdot)$ is the probability function. To solve for $P(y = 1 \mid x)$, a sigmoid function $h_\theta(x)$ can be obtained as:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \quad (11)$$

Equation 11 maps real valued output to the range $[0, 1]$ which is interpreted as the probability that the output $y = 1$.The model is trained to minimize the cost function which the log-loss is given as:

$$J(\theta) = -\left(\frac{1}{m}\right)\left[\sum_{i=1}^{m}\left((y^{(i)})\log\left((h_\theta)(x^{(i)})\right) + (1-y^{(i)})\log\left(1-\left((h_\theta)(x^{(i)})\right)\right)\right)\right] \quad (12)$$

This penalizes incorrect confident predictions harshly (example, predicting 0.99 when true class is 0). The cost function utilised in the model is convex and the convexity of the cost function must satisfy expression shown as follows:

$$\forall \theta_1, \theta_2 \in R^n, \forall \lambda \in [0, 1]:\ J(\lambda\theta_1 + (1-\lambda)\theta_2 \leq \lambda J(\theta_1) + (1-\lambda)J(\theta_2)) \quad (13)$$

Where, λ is used to denote the eigen values for the Hessian matrix, and J is used to denote the Hessian matrix. In case of functions that can be differentiated two times, a function is considered to be convex if and only if the Hessian matrix of that function is positive semi-definite. The hypothesis function is defined as:

$$h_\theta(x) = \sigma(z) \quad (14)$$

Where $z = \theta^T x$. Then the cost function is the negative likelihood for m training samples as shown in Equation 12, which is an average of individual loss term given as:

$$\ell(\theta; x^{(i)}, y^{(i)}) = -\left[y^{(i)}\log\left(h_\theta(x^{(i)})\right) + (1 - y^{(i)})\log\left(1 - h_\theta(x^{(i)})\right)\right] \quad (15)$$

This implies that if each term $\ell(\theta; x, y)$ is convex in θ, then the whole function is convex as a non-negative weighted sum of convex functions is convex. For a given data point (x, y), the sigmoid function can be defined as:

$$z = (\theta^T)(x) \Longrightarrow (h_\theta)(x) = \sigma(z) \quad (16)$$

Then the loss function becomes as shown as:

$$\ell(\theta) = -\left[y\log(\sigma(z)) + (1-y)\log(1-\sigma(z))\right] \quad (17)$$

Hessian rule can be used to compute the convexity. First, it must be established that the complement of the sigmoid function $\sigma'(z)$ is given as:

$$\sigma'(z) = \sigma(z)(1-\sigma(z)) \quad (18)$$

Hence, the derivative of the loss can be written as a standard gradient of logistic loss is given as:

$$\nabla_\theta \ell(\theta) = (\sigma(z) - y)x \quad (19)$$

Where, $\nabla_\theta(\cdot)$ is the gradient function. Furthermore, the function , $\nabla_\theta^2 \ell(\theta)$ has second derivative H given as as presented as:

$$H = \nabla_\theta^2 \ell(\theta) = \sigma(z)(1-\sigma(z)) \cdot xx^T \quad (20)$$

Notably, $(\sigma)(z)(1-\sigma(z))$ is the derivative of $\sigma(z)$. Since $z = (\theta^T)(x)$, and $\nabla z = x$, then is given as;

$$\nabla_\theta^2 \ell(\theta) = \left(\frac{d\sigma(z)}{d\theta}\right)^T \cdot \left(\frac{d\sigma(z)}{d\theta}\right) + \frac{d^2\sigma(z)}{d\theta^2} \quad (21)$$

The expansion of Equation 21 results in Equation 20. From Equation 13, the gradient descent optimization technique is used to minimize $J(\theta)$ by computing the partial derivative with respect to θ_j is given in Equation 22:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} \quad (22)$$

The update rule for the parameter vector is given as:

$$\theta : \theta - \alpha\nabla_\theta J(\theta) \quad (23)$$

Where, α is the learning rate. In this research, the function solver is set to 'bilinear'. This solver employs Newton-Raphson method (a second order optimization algorithm) which enhances convergence properties in logistic models compared to traditional algorithms. The parameter θ is updated using Newton Raphson method with the rule given as:

$$\theta^{(t+1)} = \theta^{t} - \left(H^{-1}\nabla J(\theta^{t})\right) \qquad (24)$$

Where, $\nabla J(\theta)$ is the gradient vector (first derivative), and H is the Hessian matrix (second derivative). Suppose $X \in R^{m\times n}$ is the design matrix, where each row is denoted as $x^{(i)}$, $y \in R^{m\times 1}$ is the label vector, and $h = \sigma(X\theta) \in R^{m\times 1}$ is the vector of predictions; then the gradient of the cost function $J(\theta)$ is given as:

$$\nabla J(\theta) = X^{T}(h - y) \qquad (25)$$

Where, $h = \left[h_\theta\left(x^{(1)}\right), h_\theta\left(x^{(2)}\right), \ldots, h_\theta\left(x^{(m)}\right)\right]^{T}$. The Hessian matrix is given as:

$$H = \nabla^{2} J(\theta) = X^{T}\boldsymbol{R}X \qquad (26)$$

Where $\boldsymbol{R} \in R^{m\times m}$ is a diagonal matrix, then for the second derivative of the sigmoid function is given as Equation 27 and Equation 28:

$$\boldsymbol{R}_{ii} = h^{(i)}\left(1 - h^{(i)}\right) \qquad (27)$$

$$\frac{d^{2}\sigma(z)}{dz^{2}} = \sigma(z)\left(1 - \sigma(z)\right)\left(1 - 2\sigma(z)\right) \qquad (28)$$

But since the matrix form is used in Newton Raphson method, Hessian matrix is sufficient as given in Equation 29:

$$H = \sum_{i=1}^{m} h^{(i)}\left(1 - h^{(i)}\right)x^{(i)}x^{(i)^{T}} = X^{T}RX \qquad (29)$$

Then the final Newton Raphson update rule is given as:

$$\theta^{(t+1)} = \theta^{(t)} - (X^{T}\boldsymbol{R}X)^{-1}X^{T}(h - y) \qquad (30)$$

Notably, Equation 30 is the core update formula used in Newton Raphson for logistic regression model and simulation in this work. The selected solver (liblinear) uses an optimized quasi-Newton method, particularly coordinate descent. It performs regularization, and often includes penalty, modifying the cost function to Equation 31:

$$J_{reg}(\theta) = J(\theta) + \frac{\lambda}{2}\|\theta\|^{2} \qquad (31)$$

Where $J_{reg}(\theta)$ is the regularized cost function, and λ is the regularization strength.Equation 31 adds λI to the Hessian. This regularized Hessian (shown in Equation 32) is also positive definite, improving numerical stability and convergence, is given in Equation 32:

$$H_{reg} = X^{T}\boldsymbol{R}X + \lambda I \qquad (32)$$

After training the dataset, for any input x, the binary classification prediction is made as given in Equation 33;

$$\hat{y} = \begin{cases} 1 & if\ h_\theta(x) \geq 0.5 \\ 0 & otherwise \end{cases} \qquad (33)$$

2.3 Description of the Adaptive Synthetic Sampling (ADASYN) Used for the Data Balancing

The data balancing is implemented using the Adaptive Synthetic Sampling (ADASYN) Method which builds upon the Synthetic Minority Over-sampling Technique (SMOTE) method by introducing an adaptive component. Instead of treating all minority samples equally, ADASYN focuses more on those harder to learn samples that lie in areas of high majority-class density. Given a minority sample x_i, the number of synthetic samples g_i to be generated is given as:

$$g_i = \frac{r_i}{\sum_{j=1}^{n_m} r_j} \cdot G \qquad (34)$$

Where, $r_i = \frac{\delta_i}{k}$, is the difficulty ratio with δ_i being the number of majority class samples among the k nearest neighbours of x_i, n_m is the number of minority class samples, and G is the total number of samples to generate. Also, in ADASYN, new points are generated using Equation 35.

$$x_{new} = x_i + \lambda \cdot \left(x_i^{(k)} - x_i\right) \qquad (35)$$

Particularly, ADASYN prioritizes minority samples near the decision boundary, assuming that these are more "difficult" and thus more informative. This introduces a bias toward "difficult" samples, which can potentially improve classification performance but also may increase the risk of adding noise. By the application of the ADSYN on the dataset, the number of samples of the minority class is increased to match the majority class sample size.

3. Results and discussion

3.1 The results of the data acquisition and pre-processing

The case study NASA MDP dataset has about 13 different data files containing different aspects of the software historical files. The list of all the features contained in the 12 dataset files available in the case study NASA Metrics Data Program (MDP) dataset are presented in Table 1. For each dataset (from CM1 to PC 5) in Table 1, the available feature is marked ‘Y’, while the unavailable is marked ‘N’.

In this work, the PC1.arff file which is one of the 13 files in the NASA MDP dataset is used for the description of the results obtained. Specifically, the PC1.arff file has a total of 759 labelled data records with 61 defective instances and 698 non defective instances. Data balancing was conducted using ADASYN method and the statistical profile of the original data and the synthetic data generated using the ADASYN method for the PC1.arff file is presented in Table 2. According to the results in Table 2, the original dataset has mean of 15.12 and standard deviation of 21.61 while the synthetic data has mean of 14.97 and standard deviation of 23.34. The confidence interval at 95 % confidence level shows that there is no significant difference between the mean of the original and the synthetic datasets. Hence, the dataset obtain after the data balancing using ADASYN method is a good representation of the original data record.

Table 1: The Binary Representation of Feature Availability in NASA MDP Datasets

Feature	CM 1	JM 1	KC 1	KC 3	MC 1	MC 2	MW 1	PC 1	PC 2	PC 3	PC 4	PC 5
LOC_BLANK	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
BRANCH_COUNT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
CALL_PAIRS	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
LOC_CODE_AND_COMMENT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
LOC_COMMENTS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
CONDITION_COUNT	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
CYCLOMATIC_COMPLEXITY	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
CYCLOMATIC_DENSITY	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
DECISION_COUNT	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
DECISION_DENSITY	Y	N	N	Y	N	Y	Y	Y	Y	Y	Y	N
DESIGN_COMPLEXITY	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
DESIGN_DENSITY	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
EDGE_COUNT	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
ESSENTIAL_COMPLEXITY	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ESSENTIAL_DENSITY	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
LOC_EXECUTABLE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PARAMETER_COUNT	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
GLOBAL_DATA_COMPLEXITY	N	N	N	Y	Y	Y	N	N	N	N	N	Y
GLOBAL_DATA_DENSITY	N	N	N	Y	Y	Y	N	N	N	N	N	Y
HALSTEAD_CONTENT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HALSTEAD_DIFFICULTY	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HALSTEAD_EFFORT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HALSTEAD_ERROR_EST	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HALSTEAD_LENGTH	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HALSTEAD_LEVEL	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HALSTEAD_PROG_TIME	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HALSTEAD_VOLUME	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
MAINTENANCE_SEVERITY	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
MODIFIED_CONDITION_COUNT	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
MULTIPLE_CONDITION_COUNT	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
NODE_COUNT	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
NORMALIZED_CYLOMATIC_COMPLEXITY	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
NUM_OPERANDS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
NUM_OPERATORS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
NUM_UNIQUE_OPERANDS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

NUM_UNIQUE_OPERATORS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
NUMBER_OF_LINES	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
PERCENT_COMMENTS	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
LOC_TOTAL	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Defective	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
label	N	Y	N	N	N	N	N	N	N	N	N	N

Table 2: Statistical profile of the synthetic and original data for PC1.arff in the NASA MDP Datasets

Metric	Original	ADASYN
Mean	15.12 ± 2.61	14.97 ± 2.69
Median	7.00	9.00
Std. Dev.	21.61	23.34
Variance	466.84	544.70
95% CI (Mean)	[10.90, 19.34]	[10.27, 19.67]

3.2 The results of the software bug detection using the Logistic Regression Model

The confusion matrix presented in Figure 1 represents the model's performance on the imbalanced dataset without any resampling or balancing technique. It can be observed that: True Negatives (TN): 103 instances of the negative class were correctly classified. False Positives (FP): 37 instances of the negative class were incorrectly classified as positive. False Negatives (FN): 3 instances of the positive class were incorrectly classified as negative. True Positives (TP): Only 9 instances of the positive class were correctly classified. The model exhibits a high tendency to predict the negative class correctly, which is a common issue in imbalanced datasets. While the accuracy appears moderately high, this is misleading due to the severe under-representation of the positive class. The recall score of 0.75 indicates that 75% of actual positive cases were correctly detected.

However, the precision is very low (0.196), implying a large proportion of false positives, and the F1 score (0.310) reflects poor harmonic mean between precision and recall. This performance strongly indicates the need for resampling techniques to mitigate class imbalance and improve sensitivity toward the minority class.

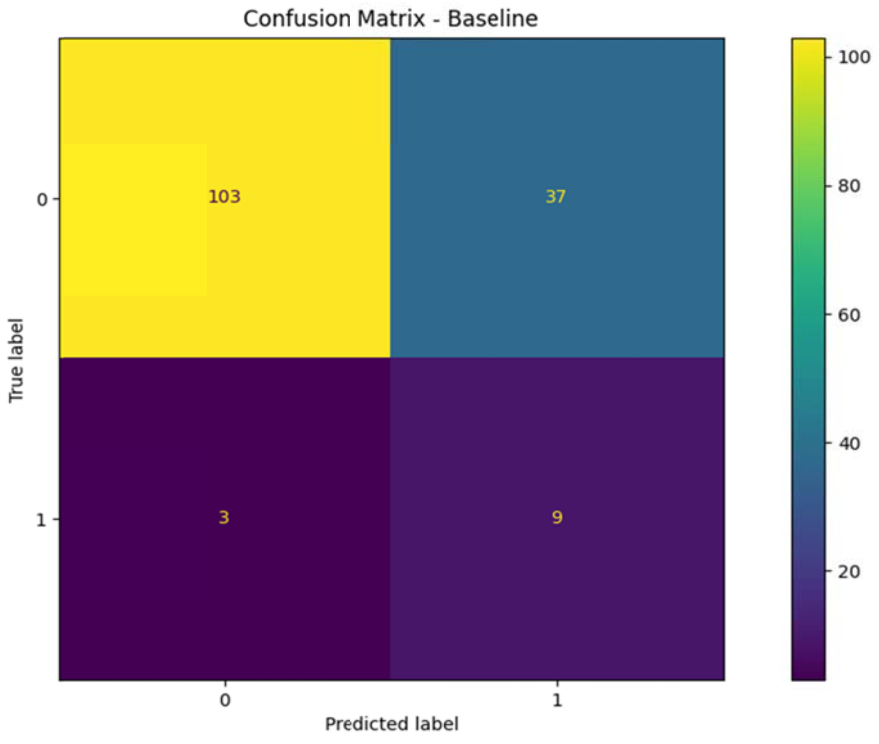

Figure 1: Confusion matrix for the baseline model case for the LRM

With ADASYN applied, the model's performance on both classes has improved as shown in Figure 2. The number of correctly classified positive instances (TP) increased significantly to 130. The false negatives increased to 11 compared to 3 in the baseline, but this is a trade-off due to the increase in sensitivity. False positives (FP) slightly decreased to 31, and true negatives (TN) increased to 109. ADASYN creates synthetic minority class samples based on the difficulty of classification. It appears to have led to a much better detection of minority instances, as seen in the sharp increase in TP. Precision improved substantially to 0.807, and recall rose to 0.922, which together boosted the F1-score to 0.861. The AUC of 0.904 suggests the model has learned to better discriminate between classes. The minor increase in training time (from about 0.019s to about 0.023s) is negligible and justifiable given the performance gains. Overall, ADASYN significantly enhanced model robustness and sensitivity toward the minority class.

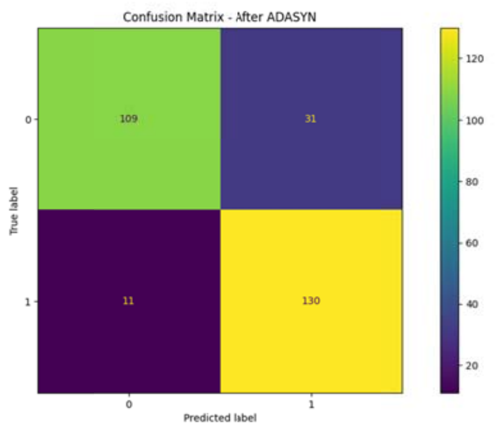

Figure 2: Confusion matrix after applying ADASYN to the LRM

3.3 The results of the software bug detection using the Random Forest Regression Model

The confusion matrix for the baseline Random Forest model as shown in Figure 3 reveals True Negatives (TN): 135 majority class instances correctly identified. False Positives (FP): 5 majority instances misclassified as minority. False Negatives (FN): 10 minority class instances misclassified as majority. True Positives (TP): Only 2 minority instances correctly classified. Despite Random Forest being an ensemble-based, relatively robust classifier, the baseline performance is heavily biased toward the majority class. While overall accuracy is high (90.13%), the metrics related to minority class detection are unsatisfactory. The recall of 0.167 implies that only about17% of defective modules were detected. The precision is also poor (0.286), resulting in a low F1-score of 0.211. The AUC of 0.850 highlights limited class separability under the class imbalance condition. This underperformance reflects Random Forest's innate bias in skewed distributions and justifies the application of resampling techniques such as ADASYN to improve sensitivity.

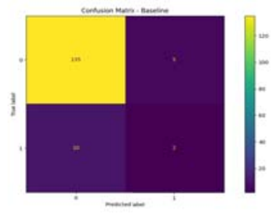

Figure 3: Confusion matrix for baseline case of the RFM

The results presented in Figure 4 reveals that the use of ADASYN leads to:

i. Substantial improvement in true positives (TP = 136).
ii. False negatives decreased from 10 to 5, meaning more defective modules are now correctly identified.
iii. Increase in false positives (from 5 to 16) as a trade-off for improved recall.
iv. Slight drop in true negatives to 124, but this is not a major concern given the model's improved recall.

ADASYN improves classification performance by generating synthetic samples adaptively around hard-to-learn minority class areas. This improved the model's recall from 0.167 to 0.965, demonstrating a substantial gain in sensitivity. Simultaneously, precision increased to 0.895, and the F1-score rose dramatically to 0.928, indicating a strong balance between precision and recall. AUC reached 0.987, showing excellent class discrimination. The training time increased to 0.725s, which reflects the additional complexity of the synthetic sampling and increased dataset size but remains within acceptable limits. ADASYN made Random Forest significantly more effective at detecting the minority class.

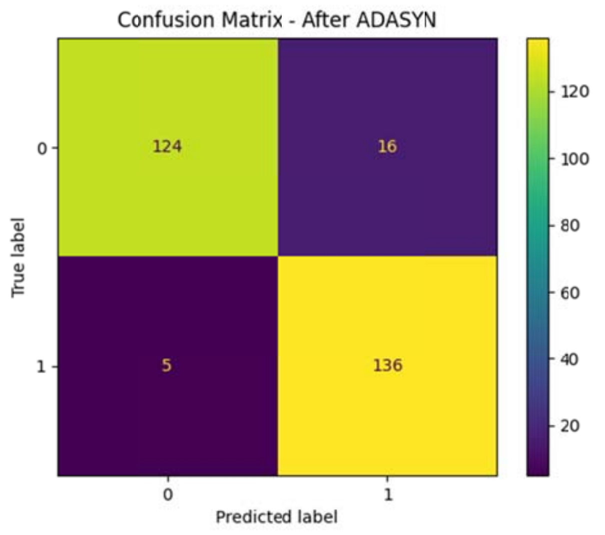

Figure 4: Confusion Matrix for the RFM after applying ADASYN data balancing

3.4 Comparison of the performance of the software Logistic Regression Model and the Random Forest Model

Comparison of the performance of the Logistic Regression model and the Random Forest model is presented in Table 3 for both the baseline case without data balancing and the case when data balancing was implemented using the ADASYN method. The results in Table 3 show that the data balancing improved the prediction accuracy of the two models.

Also, the results show that in both cases, the RFM has higher prediction accuracy than the LRM. The prediction accuracies for the baseline were 73.68 % and 90.01% for the LRM and the RFM respectively. Equally, the prediction accuracies for the case with ADASYN-based data balancing were 85.05 % and 92.53% for the LRM and the RFM respectively. Hence, in respect of the prediction accuracy, the RFM with data balancing is recommended for the software bug detection in the case study dataset.

However, the LRM has lower training time than the RFM in both cases. The training time of the RFM is about 18 times the training time of the LRM in the baseline case and about 31 times the training time of the LRM in the case with data balancing. In this wise, hybrid model that can take advantage of the low training time of the LRM and the high prediction accuracy of the RFM is recommended.

Table 3: Comparison of the performance of the Logistic Regression and the Random Forest

Evaluation Step	Accuracy	Precision	Recall	F1	AUC	Training Time
Baseline LRM	0.736842	0.195652	0.750000	0.310345	0.879762	0.019069
Baseline RFM	0.901316	0.285714	0.166667	0.210526	0.849702	0.347974
ADASYN with LRM	0.850534	0.807453	0.921986	0.860927	0.904306	0.023132
ADASYN with RFM	0.925267	0.894737	0.964539	0.928328	0.987133	0.725151

3.5 Comparison of the performance of the Random Forest Model with Some Published Related Works

The comparison of the performance of the Random Forest model with some published related works is presented in Table 4. The results showed that the RFM performed better than the published related works cited which were published between 2019 and 2024. The results indicate that with 92.53% prediction accuracy, the RFM with data balancing outperformed the previously published software detection models. However, the training time is an issue that will require further studies to minimize the training time while maintaining high prediction accuracy.

Table 4 Comparison of the performance of the Random Forest Model with Some Published Related Works

Source	Year	Method used	Accuracy (%)
Our preferred model (using ADASYN data balancing with RFM)	2025	Random Forest Model with ADASYN-based data balancing	92.527%
Our model (using RFM without data balancing)	2025	Random Forest Model without data balancing	90.13%
[20]	2019	Support vector machine and Radial Basis Function (RBF) model	90.8163 %
[21]	2019	Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM)	89.85%
[22]	2024	Random Forest Model	85.5%

4. Conclusion

Two machine learning models are presented for the prediction of software bug. The two models are the Logistic Regression Model (LRM) and the Random Forest Model (RFM). The focus in the study is to study the performance of the models with imbalanced dataset and with balanced dataset. The data balancing was implemented using Adaptive Synthetic Sampling (ADASYN). The results show that data balancing significantly improved on the software bug prediction accuracy of the two models. Also, the RFM outperformed the LRM in all the cases considered. Also, the RFM outperformed the cited related works published within 2018 and 2024. In any case, the training time of the RFM is several times higher than that of the LRM. Hence, while recommending the RFM for the software bug prediction, it is further suggested that hybridization of the LRM and the RFM can minimize the training time while maintain high prediction accuracy.

References

1. Tsybulka, K. (2024). *Enhancing code quality through automated refactoring techniques* (Doctoral dissertation, ETSI_Informatica).
2. Bhanushali, A. (2023). Ensuring Software Quality Through Effective Quality Assurance Testing: Best Practices and Case Studies. *International Journal of Advances in Scientific Research and Engineering*, *26*(1), 1-18.
3. Corradini, D., Zampieri, A., Pasqua, M., Viglianisi, E., Dallago, M., & Ceccato, M. (2022). Automated black-box testing of nominal and error scenarios in RESTful APIs. *Software Testing, Verification and Reliability*, *32*(5), e1808.
4. Wu, Y., Tao, B., Lan, K., Shen, Y., Huang, W., & Wang, F. (2022). Reliability and accuracy of dynamic navigation for zygomatic implant placement. *Clinical oral implants research*, *33*(4), 362-376.
5. Long, G., Gong, J., Fang, H., & Chen, T. (2025). Learning Software Bug Reports: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology*.
6. Ge, H., & Wu, Y. (2023). An empirical study of adoption of ChatGPT for bug fixing among professional developers. *Innovation & Technology Advances*, *1*(1), 21-29.
7. Jin, M., Shahriar, S., Tufano, M., Shi, X., Lu, S., Sundaresan, N., & Svyatkovskiy, A. (2023, November). Inferfix: End-to-end program repair with llms. In *Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering* (pp. 1646-1656).
8. Khan, R. A., Khan, S. U., Khan, H. U., & Ilyas, M. (2022). Systematic literature review on security risks and its practices in secure software development. *ieee Access*, *10*, 5456-5481.
9. Bello, R. W., & Tobi, S. J. (2023). Software bugs: detection, analysis and fixing. *Analysis and Fixing (December 12, 2023)*.
10. Singh, V. B., & Chaturvedi, K. K. (2011). Bug tracking and reliability assessment system (btras). *International Journal of Software Engineering and Its Applications*, *5*(4), 1-14.
11. Osman, H. (2017). *Empirically-Grounded Construction of Bug Prediction and Detection Tools* (Doctoral dissertation, Universität Bern).
12. Saini, S., Bhagwan, J., Rani, S., Kumar, S., Godara, S., & Chaba, Y. (2024). Early Software Bug Prediction: A Literature Review and Current Trends. *Grenze International Journal of Engineering & Technology (GIJET)*, *10*.
13. Johnson, F., Oluwatobi, O., Folorunso, O., Ojumu, A. V., & Quadri, A. (2023). Optimized ensemble machine learning model for software bugs prediction. *Innovations in Systems and Software Engineering*, *19*(1), 91-101.
14. Khalid, A., Badshah, G., Ayub, N., Shiraz, M., & Ghouse, M. (2023). Software defect prediction analysis using machine learning techniques. *Sustainability*, *15*(6), 5517.
15. Sánchez-García, Á. J., Limon, X., Dominguez-Isidro, S., Olvera-Villeda, D. J., & Perez-Arriaga, J. C. (2024). Class Balancing Approaches to Improve for Software Defect Prediction Estimations: A Comparative Study. *Programming and Computer Software*, *50*(8), 621-647.
16. Pandey, S., & Kumar, K. (2023). Software fault prediction for imbalanced data: a survey on recent developments. *Procedia Computer Science*, *218*, 1815-1824.
17. Mahajan, P., Choudhary, K., Rana, N., Kumar, R., & Deshmukh, S. (2024, July). Software Bugs Classification Using SVM, RF, DT Algorithms. In *International Conference on Data Science and Applications* (pp. 51-62). Singapore: Springer Nature Singapore.
18. Chen, Z., Ju, X., Lu, G., & Chen, X. (2022, August). Blocking bugs identification via binary relevance and logistic regression analysis. In *2022 9th international conference on dependable systems and their applications (DSA)* (pp. 335-345). IEEE.
19. Balaram, A., & Vasundra, S. (2022). Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm. *Automated Software Engineering*, *29*(1), 6.
20. Iqbal, A., Aftab, S., Ali, U., Nawaz, Z., Sana, L., Ahmad, M., & Husen, A. (2019). Performance analysis of machine learning techniques on software defect prediction using NASA datasets. *International Journal of Advanced Computer Science and Applications*, *10*(5).
21. Banga, M., Bansal, A., & Singh, A. (2019). Proposed Intelligent Software System for Early Fault Detection. *International Journal of Performability Engineering*, *15*(10), 2578.
22. ul Haq, Q. M., Arif, F., Aurangzeb, K., ul Ain, N., Khan, J. A., Rubab, S., & Anwar, M. S. (2024). Identification of software bugs by analyzing natural language-based requirements using optimized deep learning features. *Computers, Materials & Continua*, *78*(3), 4379-4397.