Universal Library of Engineering Technology Research Article

ISSN: 3064-996X

Open Access | PP: 59-68

DOI: https://doi.org/10.70315/uloap.ulete.2022.008

Data-Driven Machine Learning-Based Prediction and Performance Analysis of Software Defects for Quality Assurance

Dinesh Rajendran[1], Aniruddha Arjun Singh Singh[2], Vaibhav Maniar[3], Vetrivelan Tamilmani[4], Rami Reddy Kothamaram[5], Venkata Deepak Namburi[6]

[1]Coimbatore Institute of Technology, MSC. Software Engineering.

[2]ADP, Sr. Implementation Project Manager.

[3]Oklahoma City University, MBA / Product Management.

[4]Principal Service Architect, SAP America.

[5]California University of Management and Science, MS in Computer Information Systems.

[6]University of Central Missouri, Department of Computer Science.

Abstract

The prediction of software defects is now an indispensable part of both the quality assurance systems of the modern world providing an opportunity to detect those modules that tend to malfunction and enhance the reliability of the systems. This study offers a machine learning (ML) based framework, which uses the NASA JM1 dataset with 10,885 records and 22 features, to construct an effective prediction model. Handling of missing values, outliers, and normalization is carried out to make sure that the data is consistent. Adaptive Sequential K-Best (ASKB) is applied to discriminate the most pertinent features to improve the Minority Oversampling by Synthetic Data (MOSD), and model performance is employed to balance the classes by providing real-life defect-prone samples. The reason why a Random Forest (RF) classifier is used is its strong ability to deal with high-dimensional, complex data. The performance of model is rigorously tested utilizing Accuracy, Recall, Precision, and F1-score and yields 98.1, 98.7, 97.8 and 98.2 respectively. These pointers confirm the effectiveness of the proposed structure in the achievement of plausible predictions. Comparative analysis indicates that Trade-off of accuracy and generalization is better in Random Forest than in Naive Bayes, Neural Networks and SVM. The study will enhance the advancement of defect prediction practices, providing a scalable and explainable solution that enables proactive quality management. Further research is underway to extend this framework to hybrid and deep learning (DL) models, thereby broadening its applicability.

Keywords: *Software Defect Prediction, Machine Learning, Random Forest, MOSD, Adaptive Sequential K-Best (ASKB), Software Quality Assurance, Predictive Analytics.*

INTRODUCTION

Quality Assurance (QA) is a characteristic of modern software engineering that ensures systems meet not only technical needs but also offer reliability, maintainability, and usability in various application fields. It consists of processes[1], established practices and advanced methodologies, aimed at preventing, detecting and managing the issues in the software development life cycle [1]. Monitoring development processes, QA ensures trust in the program's products, minimizes deployment risks, and enhances adaptation to changing needs. A healthy QA practice, from the business perspective, reduces the cost of developing and maintaining the product by a significant margin, increases the efficiency of operations and also allows the product to reach the customer, in time [2][3]. At the same time, defect-free software provision will increase customer satisfaction, create user confidence, and provide an organization with a competitive advantage in the constantly challenging market.

Despite this, it remains in the centre, and software defects remain among the most enduring issues of complex systems. Error of coding is likely to result in defects [4], poor architecture, incompatibility or partial specifications of the requirements, and the result may have insignificant breakdowns up to disastrous system crashes [5][6]. A single defect in extremely sensitive sectors, such as the healthcare industry, aerospace industry, defense, and finance, may result in financial loss, negative publicity, and, worst of all, loss of human life. Some traditional defect detection methods, such as manual testing, unit testing, regression testing, and peer code reviews, are not useless [7]. However, they are often labor-intensive, subjective, and incapable of managing the size and complexity of modern software development. It is this limitation, which underscores the urgency of scalable and intelligent systems of defect prediction that may continue to satisfy the growing demands of software quality [8].

Machine Learning (ML) is a versatile and efficient, data-driven approach to better defect prediction and performance analysis within QA over the recent years [9][10]. With the

assistance of ML algorithms, it is possible to use a historical repository of defects and software metrics to identify patterns and relationships that may not be evident to human analysts automatically. The labels: Predictive models can be employed to identify modules or components as non-defect-prone or defect-prone at a very early stage of development, thereby making it feasible to take proactive measures to minimize the cost of debugging, the release cycle, and ensure on-time product delivery [11]. Furthermore, in addition to the improved accuracy of prediction, ML also enables the detailed analysis of performance, allowing organizations to evaluate the strengths and weaknesses of implemented models and optimize their QA policies accordingly [12].

Code size, cyclomatic complexity, depth of inheritance, and design features will give useful information about the probability of defects [13][14]. These quantifiable properties encapsulate the complexity and form of software systems and can be useful in predicting regions with a high probability of error. ML analysis of such features, when used properly, guides organizations in enhancing their quality assurance processes, thereby becoming more proactive, reliable, and cost-efficient.

Motivation and Contribution of the Paper

The immediate need to improve software quality through accurate defect forecasting prompted the performance of this research. Software bugs can lead to failures, financial loss, and damage to reputation, and it is essential to consider early detection as a means of successful quality assurance. Traditional approaches, such as manual inspection or rule-based methods, are often time-consuming, subjective, and inadequate for complex software systems. Data-driven machine learning techniques provide a promising alternative by automating defect prediction, identifying high-risk modules early, and enabling proactive measures to enhance software reliability and reduce development costs. This study's main contributions are outlined below:

- Utilized the NASA JM1 dataset, including 10,885 records and 22 features, to ensure a comprehensive analysis of software modules.
- Implemented stringent data pretreatment measures to enhance model reliability, such as handling missing values, detecting and removing outliers, and standardising.
- The most informative qualities were retained for defect prediction after implementing ASKB for feature selection.
- To make sure that modules prone to defects are fairly represented, we used MOSD to address class imbalance.
- Developed a Random Forest-based predictive framework capable of accurately sorting software components according to their vulnerability to bugs.
- Conducted thorough evaluations of models utilizing F1-score, recall, accuracy, and precision to guarantee practical application in software quality assurance and robust assessment.

Justification and Novelty of the Paper

The justification for this work is rooted in the urgent requirement to enhance software reliability by accurately identifying defect-prone modules before deployment. Traditional inspection and rule-based methods are often inefficient and subjective, particularly in large-scale projects. This research introduces novelty by integrating Adaptive Sequential K-Best (ASKB) for dynamic feature selection with MOSD for synthetic balancing, ensuring meaningful attributes and fair class representation. The combined framework, implemented using Random Forest, not only demonstrates improved performance but also enhances interpretability and efficiency. Unlike prior studies limited to single techniques, this work presents a holistic, data-driven approach tailored for robust software defect prediction (SDP).

Structure of the Paper

The framework of the study is presented in this section, which identifies research gaps and reviews related work. The approach, including the steps for preprocessing and designing the classifier, is detailed in Section III. Section IV describes the outcomes of the experiments and the comparisons. Section V is the final part of the work and provides directions to further research and Section VI provides the list of sources.

LITERATURE REVIEW

ML, feature optimization and class imbalance management are used in the literature on software defect prediction. Accuracy is high and scalability, generalization, real-time applicability and hybrid model integration are limited.

Manjula and Florence (2019) recommend hybrid approach based on the idea of genetic algorithms (GAs) and deep neural networks (DNNs) classification feature optimisation to predict the early software defects. The technique uses a refined version of GA able to construct chromosomes and calculate fitness functions, and a DNN strategy that uses an adaptive auto-encoder to more accurately represent software functionality. Experiments and case studies demonstrate the enhanced efficiency carried out on the PROMISE data in MATLAB. By 97.96% accuracy in PC3 dataset, 98.00% accuracy in PC4 dataset, 97.59% accuracy in CM1 dataset, 97.82% accuracy in the KC1 dataset, Compared to the existing approaches, the proposed one is superior [15]

Lingden et al. (2019) present a solution to the unequal representation of classes in datasets applied in software defect prediction (SDP). The model is comprised of a correlation feature selection (CFS) method and a modified undersampling method. The proposed model raised the precision of F1-score to 0.52-0.96 to achieve the optimal F1-score percentage of 0.96% close to 100 percent using ten open source project datasets. This strategy will address the fault prediction and reduce the processing time by enhancing

the overall outputs of the existing classification systems. It is believed that SDP is more predictable, and processing time efficient due to the emphasis that the model places on data balance [16]

Nascimento et al. (2018) discuss software testing difficulties, principles, and the lack of a comprehensive test. Using Kaizen Programming (KP), which automatically identifies high-quality nonlinear combinations of database characteristics, a method is suggested to detect problematic modules. In feature engineering, this takes the place of humans. Testers were able to uncover 216% more faults using the new features compared to random module selection, according to the study, which used a NASA open dataset with more than 9500 modules. Here we see a 1% improvement over the first features [17]

Kumar, Tirkey and Rath (2018) explored ways to identify potentially problematic modules in software by including fault prediction models into the design process of SDLC. A model was created by combining an extreme ML with different kernel approaches with twenty different kinds of software measurements. Case studies from thirty object-oriented software systems were utilised to assess the model's efficacy using a testing approach that prioritised efficiency and cost. For projects with a low (47.28%), middle (39.24%), or high (25.72%) percentage of faulty classes, the results showed that the fault prediction model was effective. They also used nine feature selection methods to weed out superfluous metrics and zero in on the best collection of source code metrics for error prediction [18]

Jacob and Raju's (2017) studied the application of machine learning and data mining methodologies for predicting software failures in aerospace systems, aiming to mitigate the consequences of software faults that can significantly endanger human lives and financial assets. The main goal of their work was to find the best feature selection and classification procedure to use in prediction modelling. They established a new hybrid feature selection methodology, which yielded a higher fault predictability of approximately 98 percent, as opposed to 82 percent. Also, the mean Matthew Coefficient (MCC) of 0.7-0.9 with an accuracy of between 86-98% was achieved due to the random subsampling process. This research notes that more research is possible on the main characteristics contributing to fault prediction with the aim of developing aerospace systems with less fault and performance [19]

Gupta and Saxena's (2017) research on software quality emphasizes the importance of software quality for users, researchers, and software developers. Using measurements, they created a model for SBPS, a software bug prediction system that can forecast class-wide issues during software validation. The Promise Software Engineering Repository is used to build the model, together with 14 relevant metrics and open-source defect datasets. When compared to other classifiers, the Logistic Regression Classifier produces the most accurate results. Separate validations of the Combined Dataset and all other validated datasets are used to train and test the model. Overall averaged accuracy of the model is 76.27%, demonstrating its effectiveness in predicting bugs in software. [20]

Table I summarizes methodologies, datasets, key findings, limitations, and future work, highlighting gaps in dataset imbalance handling, fault density sensitivity, model generalization, hybrid integration, and real-time efficiency for software defect prediction.

Table I. An overview of related research on machine learning-based software defect prediction

Author	Methodology	Dataset	Key Findings	Limitations	Future Work
Manjula & Florence (2019)	Deep Neural Network (DNN) adaptive auto-encoder and Genetic Algorithm (GA) for hybrid feature optimisation	PROMISE dataset	Hybrid approach achieved high classification accuracy (PC3: 97.96%, PC4: 98.00%, CM1: 97.59%, KC1: 97.82%,); improved feature representation enhances defect prediction	Focused only on feature optimization; limited generalization beyond selected datasets; computational cost not discussed	Extend to other datasets and explore real-time defect prediction; optimize computational efficiency of GA-DNN approach
Lingden et al. (2019)	Modified undersampling with Correlation Feature Selection (CFS)	Ten open-source project datasets	Class imbalance handling improved F1-score (0.52–0.96); reduced processing time for defect prediction	Limited to undersampling; impact of other imbalance handling methods not explored	Explore hybrid imbalance handling techniques; test on larger industrial datasets
Nascimento et al. (2018)	Kaizen Programming (KP) for automatic feature engineering and classification	NASA dataset (9500+ modules)	New features improved detection of defective modules by 216% over random selection; 1% improvement over original features	Focused on feature engineering only; improvement marginal in some cases; reliance on historical data	Investigate integration with deep learning models; real-time adaptive testing based on predictive features

Kumar, Tirkey & Rath (2018)	Elastic Learning Machine (ELM) using feature selection and different kernel approaches	30 object-oriented software systems	Fault prediction effective for low to medium faulty classes; efficiency: Low: 47.28%, Medium: 39.24%, High: 25.72%; 9 feature selection techniques evaluated	Performance decreases with high fault density; threshold-based limitation; scalability to large datasets not addressed	Develop adaptive models for varying fault distributions; optimize feature selection for large-scale projects
Jacob & Raju (2017)	Hybrid feature selection and classification; random subsampling	NASA Lunar space system software	Hybrid method improved fault prediction accuracy (~82%-98%); MCC improved (~0.7-0.9)	No single predictive model suitable for all datasets; aerospace focus may limit generalizability	Identify domain-agnostic predictive features; explore ensemble learning for consistent performance across datasets
Gupta & Saxena (2017)	Logistic Regression-based Software Bug Prediction System (SBPS) with metric validation	PROMISE datasets	Model predicted class-level bugs with overall average accuracy of 76.27%; validated 14 best metrics	Limited accuracy compared to advanced ML models; only logistic regression used; fewer datasets	Explore advanced ML and hybrid techniques; extend model for cross-project prediction and real-time QA

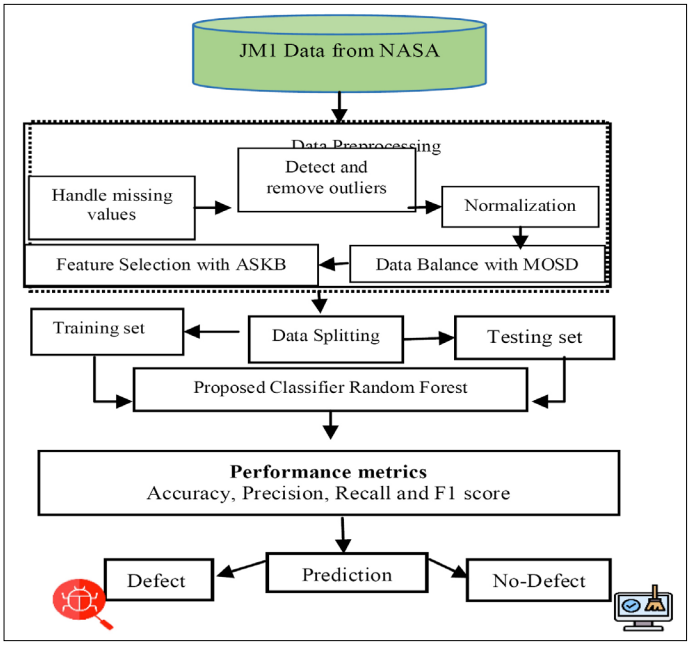

Fig 1. Proposed Methodology for SDP

METHODOLOGY

The methodology is designed to build a robust framework for predicting software defects and ensuring quality assurance, as shown in Figure 1. It employs the JM1 dataset from NASA, which consists of 10,885 records and 22 features. The first step is data preprocessing, which involves dealing with missing values, finding and removing outliers, and normalising the data so that features scale consistently. Adaptive Sequential K-Best (ASKB) is then applied for feature selection to retain the most relevant attributes, while MOSD addresses class imbalance for fair representation. To facilitate effective learning and validation, the dataset is pre-processed in an 80:20 ratio to produce a training set and a testing set. Because of its resilience and capacity to manage intricate, high-dimensional data, the Random Forest classifier was selected. As a last step in enhancing software quality, the trained model is evaluated using F1-score, recall, accuracy, and precision. It then labels modules as either non-defect or defect-prone.

Data Collection

Popular SDP datasets for real-time predictive ground code in C include the NASA-created JM1 dataset. It has 22, features, 10,885 records, and is based on McCabe and Halstead metrics including design complexity, volume, difficulty, and cyclomatic code lines, as well as Halstead metrics like essential lines of code, design complexity, and cruciality. The binary target variable signifies whether it has defects (True) or not (False). It is highly imbalanced (80.65% defective), and has been utilized to test predictive software quality models.

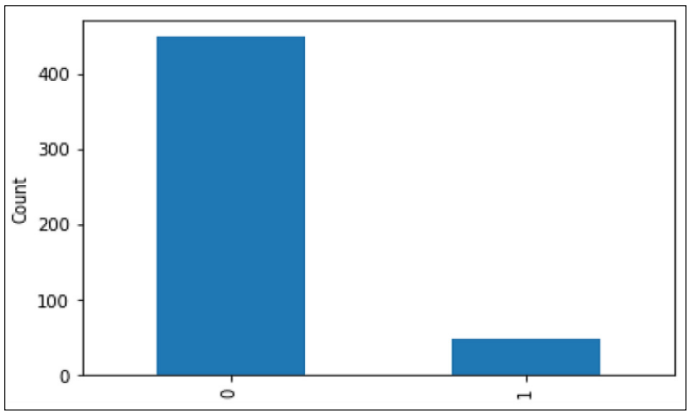

Fig 2. Class Distribution of the Dataset

The bar chart in Figure 2 is distribution of the JM1 dataset, a well-known software defect dataset, illustrates a pronounced class imbalance. The two classes, labelled '0' and '1', likely represent non-defective and defective modules, respectively. The visualization shows that class '0' dominates with over 400 instances, while class '1' has fewer than 50 instances. This corresponds to a highly imbalanced distribution, with approximately 80.65% of the modules being non-defective and only 19.35% being defect-prone. Machine learning

models may be biased towards the majority class as a result of this imbalance, missing cases from the minority class that are more likely to have flaws.

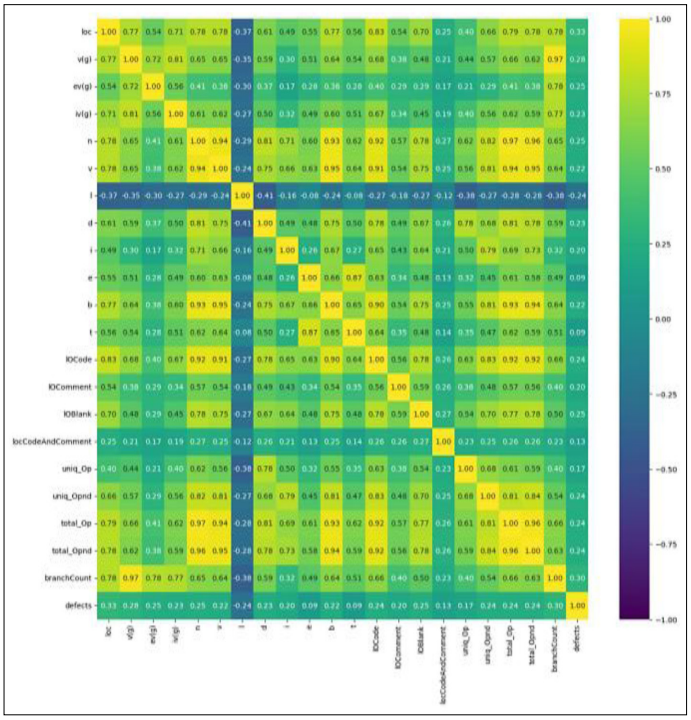

Fig 3. Correlation Heatmap of the Dataset

Figure 3 displays a heatmap displaying correlations within the JM1 dataset, illustrating the pairwise Pearson correlation coefficients among various software metrics. The value of the correlation between two attributes is represented by each cell, with a color gradient from dark blue (negative correlation) to bright yellow (positive correlation). Strong positive correlations are observed among metrics like loc, v(g), ev(g), and n. Certain metrics, such as defects and locCode, also exhibit moderate correlations with complexity-related attributes. Conversely, weak correlations are seen between some control flow metrics and defects. This visualization highlights interdependencies, aiding in feature selection and identifying potential multicollinearity in predictive modelling tasks for software defect prediction.

Data Preprocessing

An important step in data preparation for predictive modelling is cleaning and normalising raw data on software defects. Since datasets often contain inconsistencies, noise, and imbalanced distributions, pre-processing ensures improved quality, reliability, and consistency of the data. To improve dataset integrity, the following measures were applied:

- **Missing Values Handling:** Incomplete records were addressed using mean/mode imputation or KNN-based imputation, ensuring no gaps in the dataset.
- **Outlier Detection and Removal:** The Z-score method was employed to detect and eliminate abnormal values that could distort learning outcomes.

Normalization

Data normalisation is essential to prevent features with varying scales from unduly impacting the model [21]. For example, a feature like “number of commits” could have a much larger range compared to “developer activity,” leading to biased learning. To make the features comparable, they are usually transformed into the range of 0 to 1 with the help of min-max normalization. In order to make sure that every feature helps the model learn the same way, this normalisation strategy adjusts the magnitude of each feature. One is the value of the min-max normalisation Equation.(1)

$$x_{normalised} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (1)$$

This is where the original feature, denoted as x, and its minimum and maximum values, $x$min and $x$max, are defined.

Data Balancing using MOSD

Overcoming the issue of unequal class distribution in datasets of software defects Minority Oversampling by Synthetic Data (MOSD) method is used, with defect prone modules significantly underrepresented by non-defect modules. This imbalance usually makes the models used in prediction favour the majority category of predictions, thereby lowering the accuracy of determining real defects [22]. MOSD creates synthetic samples of the minority group, by nearest-neighbour-interpolation, with realistic and varied defects. With this, the distribution of the datasets is balanced and thus the classifier learns the patterns of both classes in an effective manner. The result of MOSD application, which demonstrates the transition to the balanced dataset instead of the imbalanced dataset is presented in Figure 4.

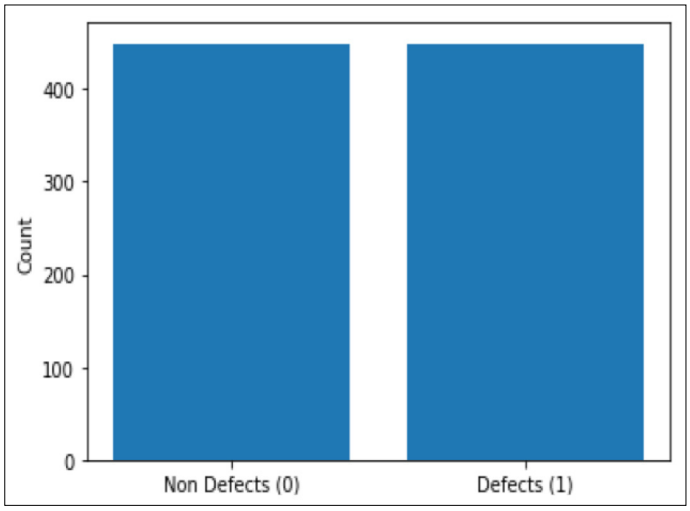

Fig 4. Dataset After Data Balancing

Feature Selection with ASKB

ASKB feature selection technique is a developed technique of choosing the most useful features in predicting software bugs. It works in an iterative manner, evaluating the contribution of every feature with statistical quantities like mutual information or chi-square and then dynamic refines

the subset [23]. ASKB lands on key features and discards redundant or noisy features by only retention; this minimizes the dimensionality and retains the crucial information. This increases training efficiency, eliminates overfitting and improves classification. In addition, ASKB makes sure that the predictive model concentrates on meaningful software metrics thus enhancing accuracy and interpretation.

Data Division

There were two sets of data utilised: A training one and a testing one. The ratio of the two sets was 80:20. This approach increases the model's generalizability, dependability, and fairness by ensuring that it is trained with sufficient data and then tested on unknown cases.

Proposed Random Forest Model

To enhance the accuracy of predictions, Bierman developed the random forest, an ensemble approach to ML which combines the result of several decision trees [24]. As shown in Figure 5, the final classification is produced by a majority voting system across all decision trees, which are each handled as a separate classifier.

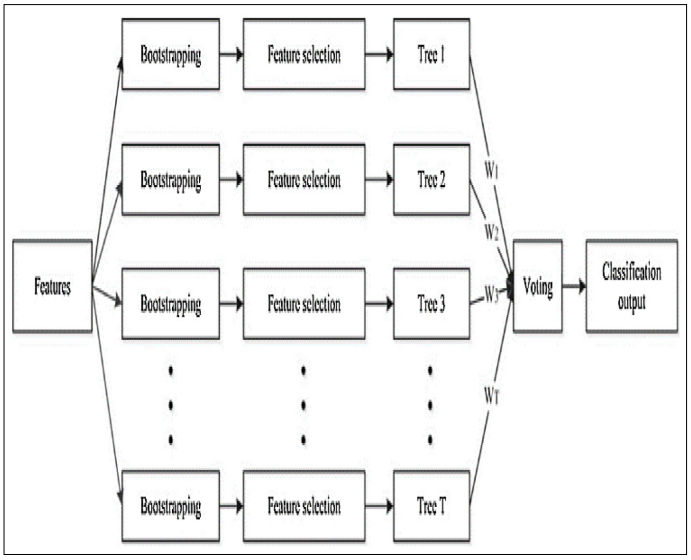

Fig 5. Structure of the Random Forest

Here, x_i∈R^M is the feature vector of size M and y_i is the associated class label, and the training dataset is comprised of N samples. Each of the T trees in a RF is trained using a bootstrap sample D_t, which is created by randomly picking N samples from the original dataset with replacement [25]. Consequently, some training samples may appear multiple times in D_t, while others may not be used at all. Instead of taking into account all M features at each divided in a tree, a randomly selected subset of m features (m<M) is used to identify the best split using an impurity measure, like the Gini index, as defined in Equation (2):

$$G(t) = 1 - \sum_{k=1}^{K} p_k^2 \quad (2)$$

Where p_k denotes the proportion of class k samples at node t, and K is the total classes. Each decision tree is permitted to grow to its max depth without pruning, ensuring that the data is fully partitioned. The last step is for all of the T trees to vote on the random forest classifier's final prediction, which can be mathematically expressed in Equation (3)

$$\hat{y} = mode\{h_t(x)|t = 1,2,\dots,T\} \quad (3)$$

Where $h_t(x)$ is what the t-th decision tree's output for input x looks like.

Performance Metrics

Error matrices, another name for confusion matrices, summarise prediction results on a classification problem and are utilised to assess the efficacy of various model approaches [26]. When dealing with a classification problem, where the output could fall into more than one category, model evaluation takes centre stage. The confusion matrix is a popular and straightforward statistic for this purpose. It measures the likelihood of four possible outcomes true negative (TN), false negative (FN), true positive (TP), and false positive (FP).

- A TP occurs when both the model's estimates and the test data are accurate.
- FN The model has obtained incorrect estimates while the test data is accurate.
- The model's estimates and test data are both incorrect; this is known as a true negative (TN).
- False positives (FPs) occur when the model's estimated true values are at odds with the test data.

Accuracy: Equation (4) expresses accuracy, often known as the categorization rate.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (4)$$

Precision: "Precision" is defined as the ratio of the number of accurately detected positive cases to the number of positive examples that were anticipated [27]. As shown in Equation (5), As decreases the value of FP, precision increases and it indicates an example labeled as positive is indeed positive.

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

Recall: Divide the sum of all actual class observations by the number of positively predicted observations to get the recall value. You can find this formula in Equation (6):

$$Recall = \frac{TP}{TP+FN} \quad (6)$$

F1-Score: When calculating accuracy, this measure considers not just recall and precision, but also FP and FN. The F-measure is determined by averaging the test's recall and precision using a weighted harmonic mean. As stated in Equation (7), this metric has an equation:

$$F1\ score = \frac{2.(Precision \cdot Recall)}{Precision+Recall} \quad (7)$$

The evaluation metrics ensure that the model is performing well by providing a thorough assessment of its accuracy, reliability, and robustness in making predictions.

RESULT ANALYSIS AND DISCUSSION

The effective detection of software defects is critical for ensuring high-quality software development, reducing maintenance costs, and improving overall system reliability. The proposed model was implemented using Python with the scikit-learn library for machine learning. This system, which runs Windows 11, has an CPU of Intel Core i7, RAM of 16 GB, and was used for the testing, which offered adequate computational power necessary to train and test it. To measure the model's efficacy, we employed Accuracy, Recall Precision, and F1-score. There is a high level of reliability in the Random Forest's defect detection performance, as shown in Table II by its 98.1% accuracy, 98.7% precision, 97.8% recall, and 98.2% F1-score. The results demonstrate that the ML algorithms like the RF may be significant in ensuring software quality by identifying faulty codes at the earliest stage and minimizing the exposure that may arise throughout the software program development process.

Table II. Performance Metrics of Random Forest Model for Software Defect Prediction

Metrics	Random Forest
Accuracy	98.1
Precision	98.7
Recall	97.8
F1-score	98.2

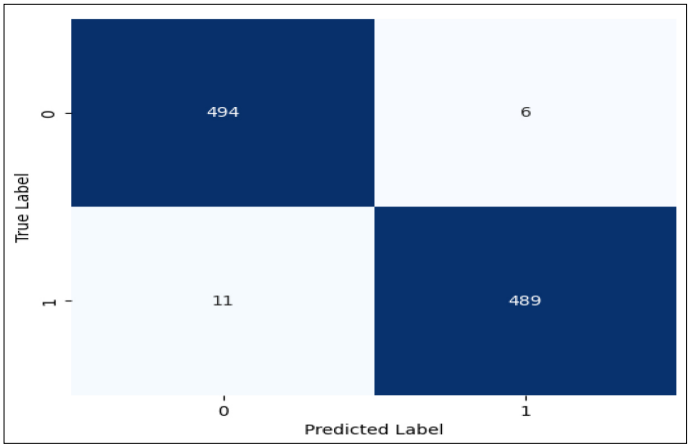

Fig 6. Confusion Matrix for the Random Forest Model

A confusion matrix, a typical visual for testing a classification model, is shown in Figure 6. There are two axes in the matrix; one axis represents the True Label (with values of 0 and 1) and the other represents the Predicted Label (with values of 0 and 1). You can see how often the model's predictions were correct in comparison to the actual results in the matrix. Class '0' (True Negatives) is shown in the top-left cell with 494 instances that were correctly identified. There are 489 examples of class '1' (True Positives) that were accurately classified in the cell located at the bottom-right. In the top-right corner, you can see six cases of class '0' being wrongly forecasted as '1' (False Positives), while in the bottom-left corner, you can see eleven cases of class '1' being wrongly predicted as '0' (False Negatives). These off-diagonal cells stand for errors. This indicates the model correctly predicted 983 out of 1000 total cases, demonstrating high accuracy.

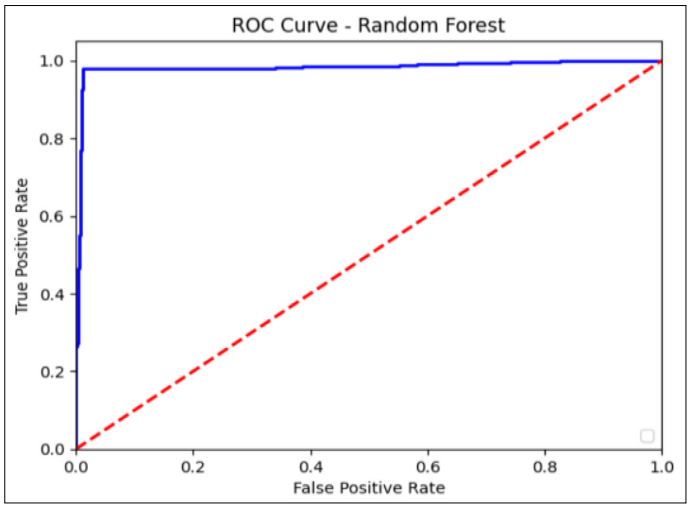

Fig 7. ROC Curve for the RF Model

The machine learning model known as the "Random Forest" classifier's Receiver Operating Characteristic (ROC) curve is displayed in Figure 7. The FP Rate (x-axis) and the TP Rate (y-axis) are plotted on the graph for various threshold settings. A solid blue line depicting the Random Forest model's performance. A dashed red line extending from (0,0) to (1,1) serves as a baseline, representing a random classifier. The blue curve hugs the top-left corner of the plot, staying well above the red dashed line. All criteria show that the model performs exceptionally well and can differentiate between positive and negative classes, as seen by its low False Positive Rate and high True Positive Rate.

Comparative Analysis

The comparison of different ML models is vital to identify the most effective approach for software defect prediction. In this work, Random Forest was evaluated against Naïve Bayes, Neural Network, and SVM. As shown in Table III, Naïve Bayes achieved good recall (94%) but low accuracy (80.39%). Neural Networks achieved greater accuracy (93.64) and poor recall (64.61), hence, missed defects. SVM was more precise (95.7% and recall (88.7%). Nonetheless, the most accurate and the best F1-score were the models based on the Random Forest, with 98.1% and 98.2, respectively, making it the most trusted tool to predict the software defects.

Table 3. Comparative Performance of Machine Learning Models for Software Defect Prediction

Models	Accuracy	Precision	Recall	F1-score
NB[28]	80.39	89	94	83
Neural network[29]	93.64	81.56	64.61	72.10
SVM[30]	86	95.7	88.7	91.8
Random Forest	98.1	98.7	97.8	98.2

The framework suggested also outlines the importance of the combination of pre-processing, feature selection, and

class balancing with a RF classifier in improving software defect prediction. The approach mitigates data imbalance, redundant attributes and noise, and therefore, the model is trained on more meaningful and representative data. Relative analysis reveals that the framework is more effective than the conventional ones, and it is effective in processing complex and high-dimensional data. In sum, the research supports the hypothesis that ML based prediction can be utilised to forecast proactive quality control, minimizing maintenance work, and permitting scalable solutions to the problem of software reliability management.

CONCLUSION & FUTURE WORK

The ensemble of a carefully designed machine learning system that includes preprocessing, feature selection, and the class balancing resulted in a great predictive accuracy on software defects. To make sure that only the most informative features were retained, ASKB was utilized, and MOSD was utilized to successfully address class imbalance. Random Forest, chosen due to its strength and scalability, gave good results with the Accuracy, Recall, Precision, and F1-score of 98.1, 98.7 and 97.8 correspondingly. It was shown that it was a superior system in relation to Naive Bayes, Neural Networks and SVM, and that LiB hypothesis can handle high dimensional and complex data, and reliably identify defect-prone modules. These results attest to the possibility of machine learning in terms of reducing the costs of tests, raising the degree of reliability, and, overall, making software quality assurance a more effective process. In the future, the work will aim at expanding the framework to include hybrid and DL models that will have the capability of capturing more intricate defect patterns. The presence of explainable AI will allow transparency, as the decision-making process will be transparent and more explainable; therefore, the approach will be more viable to software engineers. In addition, the reinforcement of the generalizability will be provided through the expansion of the evaluation to cross-project and industrial datasets. Finally, it can be incorporated into continuous integration and deployment pipelines to carry out real time monitoring which would enhance further the automation of software quality management and make predictive defect detection a continuous process of the development life cycle.

REFERENCES

1. D. D. Rao, “Multimedia Based Intelligent Content Networking for Future Internet,” in 2009 Third UKSim European Symposium on Computer Modeling and Simulation, 2009, pp. 55–59. doi: 10.1109/EMS.2009.108.
2. G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, “Software Defect Prediction via Attention-Based Recurrent Neural Network,” Sci. Program., vol. 2019, no. 1, pp. 1–14, Apr. 2019, doi: 10.1155/2019/6230953.
3. X. Zhang, K. Ben, and J. Zeng, “Cross-Entropy: A New Metric for Software Defect Prediction,” in 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, Jul. 2018, pp. 111–122. doi: 10.1109/QRS.2018.00025.
4. V. Rajavel, B. Goverdhenan, and A. V. Gomathinayagam, “Eye Gaze Pecularities Detection in Children with Autism using a Head-free cam,” Int. J. Eng. Sci. Res. Technol., vol. 5, no. 6, pp. 868–876, 2016.
5. M. Shepperd, D. Bowes, and T. Hall, “Researcher Bias: The Use of Machine Learning in Software Defect Prediction,” IEEE Trans. Softw. Eng., vol. 40, no. 6, pp. 603–616, Jun. 2014, doi: 10.1109/TSE.2014.2322358.
6. S. Dhall and A. Chug, “Software Defect Prediction Using Supervised Learning Algorithm and Unsupervised Learning Algorithm,” in Confluence 2013: The Next Generation Information Technology Summit (4th International Conference), Institution of Engineering and Technology, 2013, pp. 5.01-5.01. doi: 10.1049/cp.2013.2313.
7. A. Thapliyal, P. S. Bhagavathi, T. Arunan, and D. D. Rao, “Realizing Zones Using UPnP,” in 2009 6th IEEE Consumer Communications and Networking Conference, 2009, pp. 1–5. doi: 10.1109/CCNC.2009.4784867.
8. S. Aleem, L. F. Capretz, and F. Ahmed, “Benchmarking machine learning technologies for software defect detection,” arXiv Prepr. arXiv1506.07563, 2015.
9. S. Wang and X. Yao, “Using Class Imbalance Learning for Software Defect Prediction,” IEEE Trans. Reliab., vol. 62, no. 2, pp. 434–443, Jun. 2013, doi: 10.1109/TR.2013.2259203.
10. R. Malhotra and A. Jain, “Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality,” J. Inf. Process. Syst., vol. 8, no. 2, pp. 241–262, Jun. 2012, doi: 10.3745/JIPS.2012.8.2.241.
11. H. Foidl and M. Felderer, “Risk-based data validation in machine learning-based software systems,” in Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, New York, NY, USA: ACM, Aug. 2019, pp. 13–18. doi: 10.1145/3340482.3342743.
12. A. Balasubramanian, “Ai-Enabled Demand Response: A Framework For Smarter Energy Management,” Int. J. Core Eng. Manag., vol. 5, no. 6, pp. 96–110, 2018.
13. O. Nalbach, C. Linn, M. Derouet, and D. Werth, “Predictive quality: Towards a new understanding of quality assurance using machine learning tools,” in International conference on business information systems, 2018, pp. 30–42.
14. A. Iqbal et al., “Performance analysis of machine learning techniques on software defect prediction using NASA datasets,” Int. J. Adv. Comput. Sci. Appl., 2019, doi: 10.14569/ijacsa.2019.0100538.

15. C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," Cluster Comput., vol. 22, pp. 9847–9863, 2019.

16. P. Lingden, A. Alsadoon, P. W. C. Prasad, O. H. Alsadoon, R. S. Ali, and V. T. Q. Nguyen, "A novel modified undersampling (MUS) technique for software defect prediction," Comput. Intell., vol. 35, no. 4, pp. 1003–1020, 2019.

17. A. M. Nascimento, V. V. de Melo, L. A. V. Dias, and A. M. da Cunha, "Increasing the prediction quality of software defective modules with automatic feature engineering," in Information Technology-New Generations: 15th International Conference on Information Technology, 2018, pp. 527–535.

18. L. Kumar, A. Tirkey, and S.-K. Rath, "An effective fault prediction model developed using an extreme learning machine with various kernel methods," Front. Inf. Technol. \& Electron. Eng., vol. 19, no. 7, pp. 864–888, 2018.

19. S. Jacob and G. Raju, "Software defect prediction in large space systems through hybrid feature selection and classification.," Int. Arab J. Inf. Technol., vol. 14, no. 2, pp. 208–214, 2017.

20. D. L. Gupta and K. Saxena, "Software bug prediction using object-oriented metrics," vol. 42, no. 5, pp. 655–669, 2017.

21. A. Quality, "AI-Driven Frameworks for Efficient Software Bug Prediction and Automated Quality Assurance," vol. 7, pp. 57–66, 2019.

22. H. Alsawalqah, H. Faris, I. Aljarah, L. Alnemer, and N. Alhindawi, "Hybrid SMOTE-ensemble approach for software defect prediction," in Computer science on-line conference, 2017, pp. 355–366.

23. M. Kakkar and S. Jain, "Feature selection in software defect prediction: A comparative study," in 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence), IEEE, Jan. 2016, pp. 658–663. doi: 10.1109/CONFLUENCE.2016.7508200.

24. Y. N. Soe, P. I. Santosa, and R. Hartanto, "Software Defect Prediction Using Random Forest Algorithm," in 2018 12th South East Asian Technical University Consortium (SEATUC), IEEE, Mar. 2018, pp. 1–5. doi: 10.1109/SEATUC.2018.8788881.

25. R. Li, L. Zhou, S. Zhang, H. Liu, X. Huang, and Z. Sun, "Software Defect Prediction Based on Ensemble Learning," in Proceedings of the 2019 2nd International Conference on Data Science and Information Technology, New York, NY, USA: ACM, Jul. 2019, pp. 1–6. doi: 10.1145/3352411.3352412.

26. H. Lu, B. Cukic, and M. Culp, "Software defect prediction using semi-supervised learning with dimension reduction," in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, Sep. 2012, pp. 314–317. doi: 10.1145/2351676.2351734.

27. B. Yalciner and M. Ozdes, "Software Defect Estimation Using Machine Learning Algorithms," UBMK 2019 - Proceedings, 4th Int. Conf. Comput. Sci. Eng., no. August, pp. 487–491, 2019, doi: 10.1109/UBMK.2019.8907149.

28. G. P. Bhandari and R. Gupta, "Machine learning based software fault prediction utilizing source code metrics," in 2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS), IEEE, Oct. 2018, pp. 40–45. doi: 10.1109/CCCS.2018.8586805.

29. R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," Cluster Comput., vol. 22, no. Suppl 1, pp. 77–88, 2019.

30. P. S. Sandhu, R. Goel, A. S. Brar, J. Kaur, and S. Anand, "A model for early prediction of faults in software systems," in 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE), 2010, pp. 281–285. doi: 10.1109/ICCAE.2010.5451695.

31. Polam, R. M., Kamarthapu, B., Kakani, A. B., Nandiraju, S. K. K., Chundru, S. K., & Vangala, S. R. (2021). Big Text Data Analysis for Sentiment Classification in Product Reviews Using Advanced Large Language Models. International Journal of AI, BigData, Computational and Management Studies, 2(2), 55-65.

32. Gangineni, V. N., Tyagadurgam, M. S. V., Chalasani, R., Bhumireddy, J. R., & Penmetsa, M. (2021). Strengthening Cybersecurity Governance: The Impact of Firewalls on Risk Management. International Journal of AI, BigData, Computational and Management Studies, 2, 10-63282.

33. Pabbineedi, S., Penmetsa, M., Bhumireddy, J. R., Chalasani, R., Tyagadurgam, M. S. V., & Gangineni, V. N. (2021). An Advanced Machine Learning Models Design for Fraud Identification in Healthcare Insurance. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 2(1), 26-34.

34. Kamarthapu, B., Kakani, A. B., Nandiraju, S. K. K., Chundru, S. K., Vangala, S. R., & Polam, R. M. (2021). Advanced Machine Learning Models for Detecting and Classifying Financial Fraud in Big Data-Driven. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 2(3), 39-46.

35. Tyagadurgam, M. S. V., Gangineni, V. N., Pabbineedi, S., Penmetsa, M., Bhumireddy, J. R., & Chalasani, R. (2021). Enhancing IoT (Internet of Things) Security Through Intelligent Intrusion Detection Using ML Models. International Journal of Emerging Research in Engineering and Technology, 2(1), 27-36.

36. Vangala, S. R., Polam, R. M., Kamarthapu, B., Kakani, A. B., Nandiraju, S. K. K., & Chundru, S. K. (2021). Smart Healthcare: Machine Learning-Based Classification of Epileptic Seizure Disease Using EEG Signal Analysis. International Journal of Emerging Research in Engineering and Technology, 2(3), 61-70.

37. Kakani, A. B., Nandiraju, S. K. K., Chundru, S. K., Vangala, S. R., Polam, R. M., & Kamarthapu, B. (2021). Big Data and Predictive Analytics for Customer Retention: Exploring the Role of Machine Learning in E-Commerce. International Journal of Emerging Trends in Computer Science and Information Technology, 2(2), 26-34.

38. Penmetsa, M., Bhumireddy, J. R., Chalasani, R., Tyagadurgam, M. S. V., Gangineni, V. N., & Pabbineedi, S. (2021). Next-Generation Cybersecurity: The Role of AI and Quantum Computing in Threat Detection. International Journal of Emerging Trends in Computer Science and Information Technology, 2(4), 54-61.

39. Polu, A. R., Vattikonda, N., Gupta, A., Patchipulusu, H., Buddula, D. V. K. R., & Narra, B. (2021). Enhancing Marketing Analytics in Online Retailing through Machine Learning Classification Techniques. Available at SSRN 5297803.

40. Polu, A. R., Buddula, D. V. K. R., Narra, B., Gupta, A., Vattikonda, N., & Patchipulusu, H. (2021). Evolution of AI in Software Development and Cybersecurity: Unifying Automation, Innovation, and Protection in the Digital Age. Available at SSRN 5266517.

41. Polu, A. R., Vattikonda, N., Buddula, D. V. K. R., Narra, B., Patchipulusu, H., & Gupta, A. (2021). Integrating AI-Based Sentiment Analysis With Social Media Data For Enhanced Marketing Insights. Available at SSRN 5266555.

42. Buddula, D. V. K. R., Patchipulusu, H. H. S., Polu, A. R., Vattikonda, N., & Gupta, A. K. (2021). INTEGRATING AI-BASED SENTIMENT ANALYSIS WITH SOCIAL MEDIA DATA FOR ENHANCED MARKETING INSIGHTS. Journal Homepage: http://www. ijesm. co. in, 10(2).

43. Gupta, A. K., Buddula, D. V. K. R., Patchipulusu, H. H. S., Polu, A. R., Narra, B., & Vattikonda, N. (2021). An Analysis of Crime Prediction and Classification Using Data Mining Techniques.

Citation: Dinesh Rajendran, Aniruddha Arjun Singh Singh, et al., "Data-Driven Machine Learning-Based Prediction and Performance Analysis of Software Defects for Quality Assurance", Universal Library of Engineering Technology, 2022; 59-68. DOI: https://doi.org/10.70315/uloap.ulete.2022.008.