# Software Bug Prediction Using Supervised Machine Learning Algorithms

[1]Sushma, [2]Dr. Prasad Gr

[1]Student, [2]Professor
M.Tech in CSE
Department of Computer Science and Engineering Bangalore India

***Abstract*- The research focuses on predicting software defects to enhance industrial success by providing measurable outcomes for development teams. Identifying defective code areas aids developers in bug pinpointing and optimizing testing efforts. Early detection depends on achieving a high percentage of accurate classification, which is crucial. Although software-defected data sets are large, they are only partially recognized and supported. In contrast to previous research that utilized the Weka simulation tool, In this paper, the machine learning techniques of Logistic Regression, Support Vector Machine (SVM), and Random Forest (RF) are proposed. The systematic analysis measures parameters like confusion, precision, recall, and recognition accuracy, comparing them to existing methods. According to the results, Random Forest had a remarkable accuracy of 98.85% while SVM had an accuracy of 97.75%. However, Logistic Regression lagged behind with 64% accuracy.**

***Keywords*- Software defects, Logistic Regression, Support Vector Machine (SVM), Random Forest (RF)**

## I.Introduction

The ability to forecast software bugs is essential in the world of software engineering as it endeavors to detect potential errors and defects in software before they materialize. The existence of bugs in software can lead to catastrophic consequences, such as system failures, security vulnerabilities, and user dissatisfaction, resulting in expensive rework, ongoing maintenance, and damage to the reputation of software products and their developers. However, by proactively identifying and addressing software bugs early in the development cycle, the overall software quality can be enhanced, and the associated costs can be mitigated.

The process of software bug prediction involves a comprehensive analysis of historical data and software metrics to unveil patterns and trends that can be leveraged to anticipate the likelihood of future bugs. A range of techniques has been developed to tackle software bug prediction, encompassing statistical models, machine learning algorithms, and data mining methods. These approaches rely on diverse types of software metrics, such as code complexity, code churn (the frequency of code changes), and code coverage (the proportion of code exercised during testing), to pinpoint potential bugs and prioritize areas requiring thorough testing and debugging. Software defect risk reduction, software quality improvement, and software testing & debugging process efficiency are the main goals of software bug prediction. In recent years, this domain of research has garnered significant attention, driven by the ever-increasing complexity of software systems and the pressing need for more streamlined and effective software development practices.

As the software industry continues to evolve and innovate, software bug prediction is poised to grow further, with ongoing research dedicated to refining the accuracy and efficiency of bug prediction techniques. This introductory overview provides valuable insights into the significance of software bug prediction and offers a glimpse into the current state of research in this dynamic field. As the quest for improved software reliability continues, advancements in bug prediction methodologies hold the potential to revolutionize software development practices and safeguard against the potentially devastating impact of software defects.

## II.Related Work

Jalaj Pachouly et al., [1provided a program that makes it easier to create high-quality datasets, covering the gathering of raw data, data pre-processing, validation, & prediction using particular machine learning algorithms. The tool, SDPTool, dynamically gathers information from a GitHub repository and has settings for machine learning, data pre-processing, and various defect predictions. Nasraldeen Khleel et al. in [2] developed an SDP approach using convolutional neural networks (CNN) along with gated recurrent units (GRU), coupled with artificial minority oversampling technique using Tomek link (SMOTE Tomek). Their model's performance was evaluated on benchmark datasets from the PROMISE repository, and results were compared using the accuracy, precision, recall, F-measure, area under the receiver's operating characteristic (ROC) curve (AUC), area under the precision-recall curve (AUCPR), and mean square error (MSE). According to a study by MOHAMMED AKOUR et al., the performance of the deep learning algorithms Multi-layer perceptrons (MLPs) and Convolutional Neural Network (CNN) can be enhanced by modifying the settings. The experiments showed that parameter modification significantly impacted prediction performance, resulting in higher rates compared to traditional machine learning algorithms. JIE ZHANG et al., [4] addressed software defect prediction with a novel framework, paradigm for cross-version data selection (CDS). Their proposed CDS model outperformed three baseline techniques and a cutting-edge approach in several performance measures, using a Weighted Sampling Model (WSM) for fresh data and a Clustering-based Multi-Version Classifier (CMVC) on older files. In order to quantify the likelihood that a class would contain bugs, Xin Du et al. [5] introduced an enhanced effort-aware bug prediction method they named Core Bug. This method uses the weighted direct class dependency network (WDCDN) as well as modified k-core decomposition. Experimental outcomes on Java applications proved Core Bug's superiority to cutting-edge methods. FAIZA KHAN et al.,.[6] proposed a methodology for predicting software bugs that combines the Artificial Immune Network (AIN) in hyper-parameter optimisation with machine learning classifiers. The model fared better than cutting-edge classifiers, demonstrating its potency in bug prediction. [7] A software fault prediction technique using program slice & deep learning was put out by Junfeng Tian et al. When compared to the Tree-LSTM method, their approach significantly improved the F1-measure for both within-project & defect prediction across projects.[8] Algorithms for machine learning, such as Logistic Regression, Decision Tree, and Random Forest Adaboost, & XGBoost were compared by Aashish Gupta et al. before a new XGBoost model with modified parameters was suggested. For a number of datasets, their model fared better than cutting-edge models.[9] In order to find association rules for useful predictors of faults, Nasir Mahmood et al. suggested a fault prediction strategy using the Apriori algorithm superior software. The proposed rules showed promising results in an experimental study with different industrial projects. Xuan Zhou et al., [10] developed LSTM-BT, To forecast defect-proneness based on semantic & syntactic data retrieved from source codes, A bidirectional and tree-structured long short-term memory network was created.[11] ZHIGANG LIU et al. examined the use of extreme learning machines (ELM) for failures in software prediction & proposed a weighted regularization ELM variant in order to effectively manage unbalanced data. Santosh Singh Rathore et al. [12] presented generative the oversampling approaches leveraging generative adversarial network (GAN) to address the problem of imbalanced data in software failure prediction models. [13] A new SDP model was proposed by Kechao Wang et al. with the goal of increasing prediction accuracy that integrates the support vector machine algorithm with the minimal absolute value compression & selection approach.

Yu Qu et al., [14] used node2defect, an error prediction model integrating embedded vectors with conventional software engineering metrics, to perform a thorough empirical study comparing network embedding methods in bug prediction. Ebubeogu Amarachukwu Felix et al., [15] presented a method for software defect prediction using predictor variables derived from defect acceleration. Their integrated machine learning approach showed promising results across ten different datasets from the PROMISE repository.

### Machine Learning

"Machine learning" is a branch of artificial intelligence that focuses on developing algorithms that enable computers to learn from data and improve over time at specific tasks without needing to be manually programmed. It combines reinforcement learning for environmental decision-making with unsupervised learning to find patterns in unlabeled data and supervised learning to learn from labeled data. Regression, classification, clustering, & dimensionality reduction are all examples of machine learning techniques, as are neural networks modeled after the human brain. Applications span natural language processing, computer vision, recommendation systems, finance, healthcare, and more. Data preparation, feature engineering, hyper parameter tuning, and model evaluation are crucial for success. As data and computational resources grow, machine learning continues to advance, impacting various aspects of daily life with its ability to solve complex problems and deliver valuable insights.

## III.Proposed Method

The research proposes a system for predicting software defects to improve overall software quality. For the purpose of early defect detection, It employs the Logistic Regression, Support Vector Machine (SVM), and Random Forest (RF) machine learning algorithms. The system gathers and preprocesses data sets that have been tainted by software. A systematic analysis measures parameters like confusion, precision, recall, and recognition accuracy, comparing them to existing methods. The findings show that Random Forest achieved an accuracy of 98.85%, SVM attained 97.75%, while Logistic Regression lagged with 64% accuracy.

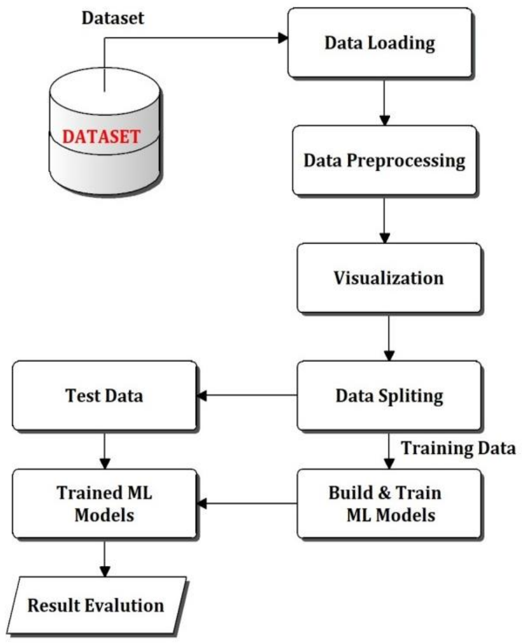


*Fig 1: Proposed System Architecture*

*1.* **Logistic regression**

For binary classification tasks, the statistical & machine learning algorithm logistic regression is frequently utilized. It uses the logistic (sigmoid) functions to compute probabilities and models the link between input features as well as binary outcomes. The logistic function converts an input feature and coefficient combination linearly into a number between 0 and 1, which represents the likelihood of the class being positive. Finding the best coefficients during training increases the chance of the results that are observed. The effectiveness, interpretability, and simplicity of logistic regression are highly regarded. It suits various applications, including medical diagnosis, credit risk assessment, and spam detection.

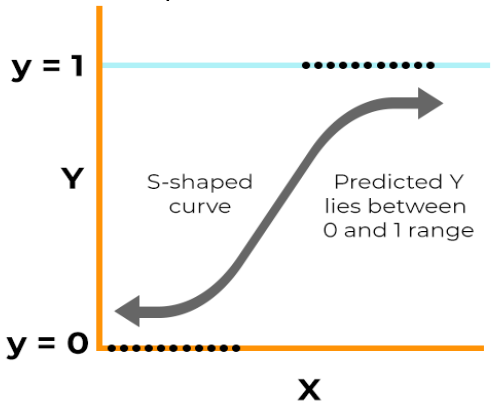


*Fig 2: the Logistic regression Architecture*

***2.*** **Support Vector Classifier**

The Support Vector Classifier (SVC), also known as the Support Vector Machine (SVM), is a powerful supervised machine learning method for classification tasks. It seeks to identify the ideal hyperplane for classifying data points into groups while maximizing the distance between them. The margin denotes the separation between the nearest data points, or support vectors, and the hyperplane. By applying the kernel method, SVC may handle non-linearly separable data by converting it to a higher-dimensional space. During training, SVC minimizes a cost function to reduce misclassification while maximizing the margin. It is widely used in image classification, text categorization, and bioinformatics due to its ability to handle complex decision boundaries.

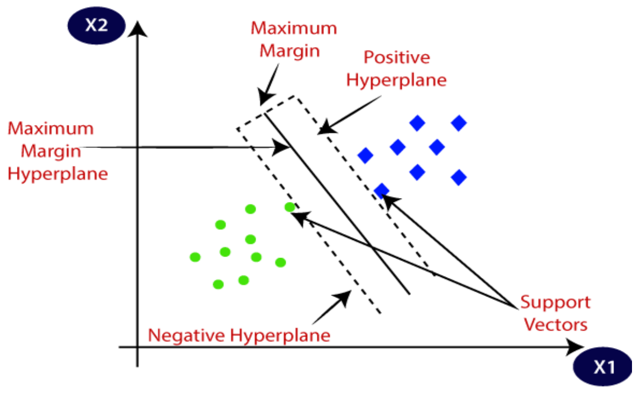


**Fig 3: The Support Vector classifier Architecture**

**3. Random Forest Classifier**

A flexible ensemble learning technique used for classification & regression applications is called Random Forest. To produce reliable and accurate forecasts, it combines a number of decision trees. The approach uses a voting mechanism for classifying or an averaging mechanism for regression by splitting the training data into smaller subsets & learning tree's of decisions on each subset. Notably, Random Forest exhibits robustness and mitigates over fitting risks, while also offering insights into feature importance. Its scalability allows it to handle large datasets efficiently, and its parallelizability is an advantage for distributed computing environments. Due to its effectiveness, Random Forest finds applications in various fields, making it a valuable tool in the realm of machine learning.

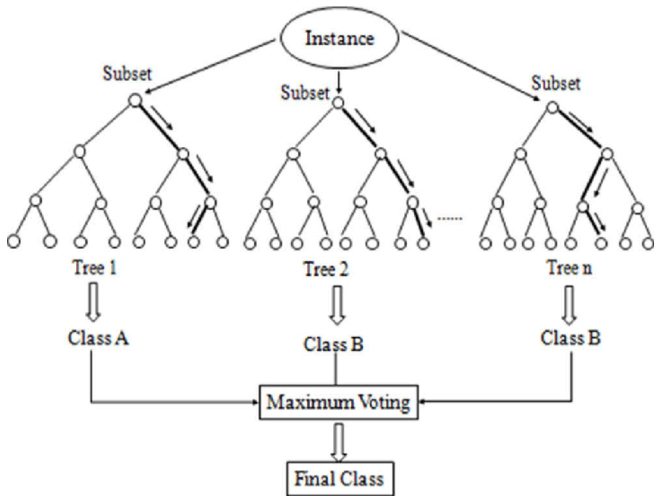
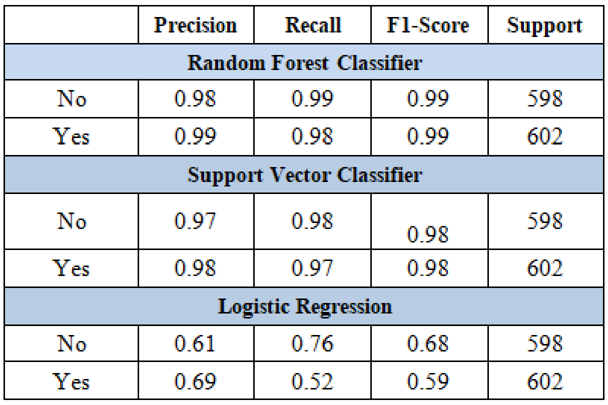


*Fig 4: The Random Forest Classifier Architecture*

## IV.Result Analysis

Using analytical criteria such recall, precision, F1-score, as well as contribution for each target class, the classification report provides a thorough assessment of the effectiveness of the classification model. These metrics summarize the model's capability in correctly classifying different groups. Assessing the report is crucial in determining the model's effectiveness, accuracy, and ability to classify instances across multiple classes. Analyzing the report provides valuable insights into the model's performance, enabling informed decisions about its suitability for a specific classification task.

**Precision:** The number of of the expected positive examples are actually significant is measured by precision.
***Precision = True Positives / (True Positives + False Positives)***
**Recall:** Recall measures the proportion of actual positive events that the model accurately detected.
***True Positives / (True Positives + False Negatives)***
**F1-score:** In unbalanced datasets, F1-score offers equilibrium between precision & recall.
***F1-score = 2 * (Precision * Recall) / (Precision + Recall)***
**Support:** Support shows how many examples there are of every category in the dataset.
The classification report aids in comprehending the model's performance for each class individually and collectively.



| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| **Random Forest Classifier** | | | | |
| No | 0.98 | 0.99 | 0.99 | 598 |
| Yes | 0.99 | 0.98 | 0.99 | 602 |
| **Support Vector Classifier** | | | | |
| No | 0.97 | 0.98 | 0.98 | 598 |
| Yes | 0.98 | 0.97 | 0.98 | 602 |
| **Logistic Regression** | | | | |
| No | 0.61 | 0.76 | 0.68 | 598 |
| Yes | 0.69 | 0.52 | 0.59 | 602 |

***Table 1: Performance Metrics Comparison***

The offered Logistic Regression, Support Vector Classifiers, & Random Forest Classifier performance metrics are shown in Table 1. Precision, Recall, F1-Score, & Support are the measurement measures that are employed. Precision measures the classifier's accuracy in correctly identifying positive instances among the predicted positives for each class. For the Random Forest Classifier, it achieves high precision values of 0.98 and 0.99 for classes "No" and "Yes," respectively. Recall, often referred to as Sensitivity, measures how well a classifier can pick out positive examples among all of the real positives for a given class. For the classifications "No" and "Yes," the Random Forest Classifier achieves recall scores of 0.99 and 0.98, respectively.The F1-Score, which offers a fair evaluation, is both the Precision and Recall harmonic means. The Random Forest Classifier's F1-Score for both classes is 0.99, which indicates overall strong performance.
The total amount of copies for each class is indicated in the Support column. In the Random Forest Classifier, there are a total of 598 instances of "No" & 602 instances of "Yes."

**Confusion matrix**
Using a square matrix known as the confusion matrix, a classification model's performance can be evaluated by contrasting its forecasts with the actual outcomes from the dataset. The confusion matrix consists of rows representing the true or actual classes and columns representing the predicted classes. By analyzing the values in the confusion matrix, we can determine how well the model is correctly classifying instances into different classes and gain insights into its overall performance.

**A. Random Forest Classifier**

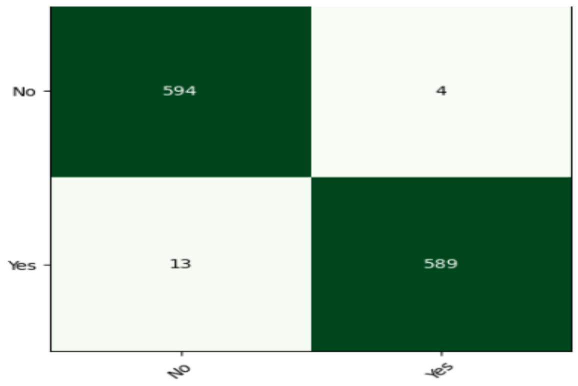


***Fig 5: Confusion Matrix for Random Forest Classifier***

The Random Forest Classifier model's confusion matrix is shown in Figure 5, which demonstrates how well it performs. The model achieves an impressive accuracy of 98.58%.

### B. Support Vector Classifier

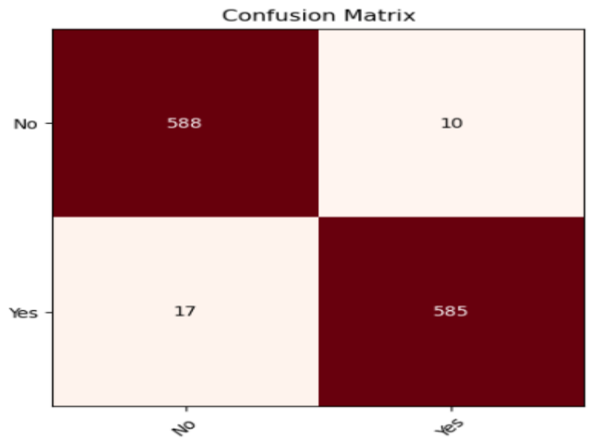


*Fig 6: Confusion Matrix for Support Vector Classifier*

Figure 6 uses a confusion chart to show the Support Vector Classifier model's performance showcasing its accuracy rate of 97.75 %.

### C. Logistic Regression

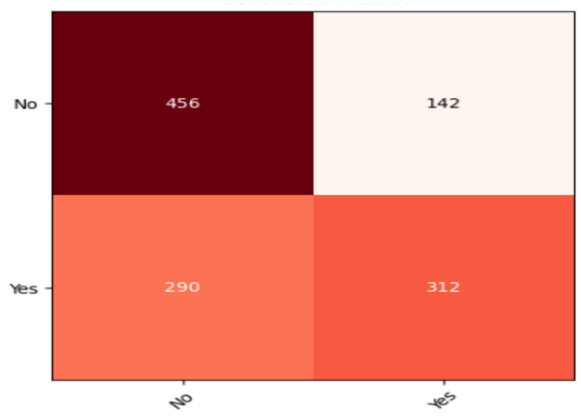


*Fig 7: Confusion Matrix for Logistic Regression Classifier*

The performance of the Logistic Regression Classifier model is illustrated in Figure 7 through a confusion chart, showcasing its accuracy rate of 64%.

## V.Conclusion

The research focuses on using machine learning models for enhance defect detection, classification, and software issue prediction. Defect detection and testing effectiveness are improved through the suggested models (Logistic Regression, Support Vector Machine, and Random Forest). The study shows that a Random Forest approach attained a remarkable accuracy of 98.85%, followed by SVM with 97.75%. However, Logistic Regression performed less effectively, attaining only 64% accuracy.

**REFERENCES:**

[1] Abdu, Ahmed, et al. "Deep Learning-Based Software Defect Prediction via Semantic Key Features of Source Code—Systematic Survey." Mathematics 10.17 (2022): 3120.
[2] Khleel, Nasraldeen, and KárolyNehéz. "A Novel Approach for Software Defect Prediction using CNN and GRU Based on SMOTE Tomek Method." (2022).
[3] Al Qasem, Osama, Mohammed Akour, and Mamdouh Alenezi. "The influence of deep learning algorithms factors.in software fault prediction." IEEE Access 8 (2020): 63945-63960.
[4] Zhang, Jie, et al. "CDS: A cross–version software defect prediction model with data selection." IEEE Access 8 (2020): 110059-110072.
[5] Du, Xin, et al. "CoreBug: Improving effort-aware bug prediction in software systems using generalized k-core decomposition in class dependency networks." Axioms 11.5 (2022): 205.
[6] Khan, Faiza, et al. "Hyper-parameter optimization of classifiers, using an artificial immune network and its application to software bug prediction." IEEE Access 8 (2020): 20954-20964.
[7] Tian, Junfeng, and Yongqing Tian. "A model based on program slice and deep learning for software defect prediction." 2020 29th International Conference on Computer Communications and Networks (ICCCN). IEEE, 2020.
[8] Gupta, Aashish, et al. "Novel xgboost tuned machine learning model for software bug prediction." 2020 international conference on intelligent engineering and management (ICIEM). IEEE, 2020.

[9] Mahmood, Nasir, et al. "Mining software repository for cleaning bugs using data mining technique." Computers, Materials & Continua 69.1 (2021): 873-893.
[10] Zhou, Xuan, and Lu. "Defect prediction via LSTM based on sequence and tree structure." 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2020.
[11] Bal, Pravas Ranjan, and Sandeep Kumar. "WR-ELM: Weighted regularization extreme learning machine for imbalance learning in software fault prediction." IEEE Transactions on Reliability 69.4 (2020): 1355-1375.
[12] Rathore, Santosh Singh, et al. "Generative Oversampling Methods for Handling Imbalanced Data in Software Fault Prediction." IEEE Transactions on Reliability 71.2 (2022): 747-762.
[13] Wang, Kechao, et al. "Software defect prediction model based on LASSO–SVM." Neural Computing and Applications 33 (2021): 8249-8259.
[14] Qu, Yu, and Heng Yin. "Evaluating network embedding techniques' performances in software bug prediction." Empirical Software Engineering 26 (2021): 1-44.
[15] Felix, EbubeoguAmarachukwu, and Sai Peck Lee. "Integrated approach to software defect prediction." IEEE Access 5 (2017): 21524-21547.