Journal of Science, Computing and Engineering Research (JSCER)
Volume-8, Issue-6, June 2025.
DOI: *https://doi.org/10.46379/jscer.2025.080603*

Software Defects Prediction Using Machine Learning Algorithm

[1]R. Anusuya, [2]Sadhana Yadav

[1]Professor, Department of CSE, Modern Institute of Technology & Research Centre,Alwar,Rajasthan.

[2]PG Student, Department of CSE, Modern Institute of Technology & Research Centre, Alwar,Rajasthan.

Article Information

Received : 24 June 2025
Revised : 26 June 2025
Accepted : 28 June 2025
Published : 30 June 2025

Corresponding Author:
Sadhana Yadav

Abstract—Software quality is one of the essential aspects of a software. With increasing demand, software designs are becoming more complex, increasing the probability of software defects. Testers improve the quality of software by fixing defects. Hence the analysis of defects significantly improves software quality. The complexity of software also results in a higher number of defects, and thus manual detection can become a very time-consuming process. The K Neighbors Classifier (KNN) exhibited high accuracy (0.99) and well- balanced precision (0.98), recall (0.96), and F1-Score (0.93), indicating robust predictive capabilities. The Ensemble model, a kin to the Decision Tree, demonstrated exceptional accuracy (0.99) and perfect precision, recall, and F1-Score, affirming its effectiveness in software defect prediction.

Keywords: Software Quality, Software Defects, Software Testing, Machine Learning, NASA Promise dataset

Citation: R. Anusuya, Sadhana Yadav, "Software Defects Prediction Using Machine Learning Algorithm", Journal of Science, Computing and Engineering Research, 8(05), May 2025.

I. INTRODUCTION

With growing demand and technology, the software industry is rapidly evolving. Since humans do most of the software development, defects will inevitably occur. In general, defects can be defined as undesired or unacceptable deviations in software documents, programs, and data [1].Defects may exist in requirements analysis because the product manager misinterprets the customer's needs, and as a result, this defect will also carry on to the system design phase. Defects may also occur in the code due to inexperienced coders. Defects significantly impact software quality, such as increased software maintenance costs, especially in healthcare, and aerospace software defects can have serious consequences. If the fault is detected after deployment, it causes an overhead on the development team as they need to re-design some software modules, which increases the development costs. Defects are nightmares for reputed

II. CONCEPTS AND OVERVIEW OF SOFTWARE DEFECTS

A. Concept of software defects

Since analysts often can't distinguish between software defects and programming faults, errors, and failure, this article utilizes IEEE729- 1983(Standard Glossary of Software Engineering Terminology) to characterize defects as, From the inside of the product, the defects are mistakes and errors in the maintenance or development of the product item. From an external perspective, a defect is the violation or failure of the framework/system to accomplish specific capacities [3,4].The description of the concepts that are easily mistaken with defects is as follows

1. Fault: The software doesn't perform according to the client's expectations and runs in an unsuitable internal state. We can view it as a defect that can prompt software errors and is regarded as dynamic behavior.

2. Failure: It refers to the outputs that the software generates at runtime, which the client doesn't accept. For instance, if the execution capacity is lost, and the client's capabilities are not met, the framework can't meet the fixed asset's execution necessities.

3. Error: It is introduced by individuals and changed over into faults under specific conditions. It exists in the whole software life cycle, including error information in the software design, data structure, code, requirements analysis, and other carriers [5].

The quality of software relies upon the number of defects. An excessive number of defects lead to reduced client satisfaction, consuming organization assets and expenses, and slower testing. To spare the costs, improving test productivity is critical to managing defects.

B. Main research directions of software defects

1. Software defect management

Defect management mainly refers to the collection ,statistics, and useful recording of defects. To improve management productivity, engineers have designed many

robotized defects management devices. At present, the industry's commonly used tools are mainly JIRA dispatched by Atlassian and Bugzilla, an open-source bug tracking framework provided by Mozilla. The set two tools record the transactions, attributes, statistical information of defects, and don't have a more profound investigation and explicit grouping of defects. Defect analysis and classification are significant segments of defect management. Therefore, examination and arrangement of deformities require further exploration of the data recorded in JIRA and Bugzilla.

2. Software defect analysis

Software designers regularly use defect analysis to better access programming quality and development quality. Software defect analysis is a strategy for characterizing imperfections and mining the reasons for defects. The motivation behind software defect analysis is to enable analysts to find, locate, evaluate, and improve test efficiency. The defects analysis methods are mostly partitioned into qualitative analysis, quantitative analysis, and attribute analysis[4]. Qualitative analysis strategies mostly incorporate Root Cause Analysis (RCA) and Software Fault Tree Analysis (SFTA). Attribute analysis is commonly partitioned into single attribute analysis and multi-attribute analysis.

3. Software defect classification

Software defects are different and complex. A more clear grouping and conglomeration of deficiencies can assist programmers with assessing programming quality, improve analyzers' work productivity, and decrease the trouble of analysis. Classification is likewise useful to suggest repair techniques and reuse test cases [2]. It can comprehend the distribution of defects as per the Classification and analysis results, prevents frequently occurring software defects, extraordinarily improve the software development cycle, and in this way, improve the quality of software[6,7]. In this way, software defects classification is a significant piece of software defect analysis. The outcomes of defect classification directly influences the defect analysis process, so defect classification has great significance. Up until now, software defect classification can be partitioned into manual classification and programmed/automatic classification.

1. Manual classification of software defects: Software defect manual classification implies that examiners utilize their insight to group defects into various classes. To start with, the researchers set the ideal classification of defects. They then discover the fault and type match the defect based on experience. None the less, the classification cycle of this strategy is quite complicated and requires a large team. Because of restricted human energy and memory, a lot of data analysis will bring about a lower classification speed, far lower than the computer, and consequently burn-through a great deal of time and assets.

III. AUTOMATIC CLASSIFICATION OF SOFTWARE

Defects: To reduce development costs and improve development productivity, individuals are more inclined to use computers to automatically classify defects. Specialists are attempting to locate a straight forward method to classify defects, and the ascent of AI and machine learning has made the automatic classification of defects a hotspot for industrial research.

IV. RESEARCH METHODOLOGY

The proposed methodology for software failure prediction employs a systematic approach, encompassing critical stages essential for comprehensive data analysis and model performance assessment.

A. DATA INFORMATION

The JM1 dataset, a component of the PROMISE repository, is designed for software defect prediction and has been made publicly available by NASA and the NASA Metrics Data Program. Naive Baye's, derived from JM1, has exhibited promising performance, out performing J48 for defect detection, and the dataset has highlighted the nuanced relationship between accuracy and the effectiveness of defect detectors. The JM1 dataset comprises 10,885 instances, each characterized by 22 attributes.

Notably, there are no missing attributes in the dataset. The class distribution indicates that 19.35% of instances are labeled as "false" (modules without reported defects), while 80.65% are labeled as "true" (modules with one or more reported defects).

Figure: data frame

B. DATA PRE-PROCESSING

In the preprocessing pipeline for software failure prediction using machine learning, several critical steps are undertaken to refine textual data and optimize it for analysis, the standard scaler technique is applied to normalize and scale the data, enhancing its numerical stability. Subsequently, the text is transformed to lowercase to ensure consistency and mitigate case- related variations. To facilitate readability and semantic analysis, punctuation signals are then systematically removed from the text. The

final preprocessing step involves lemmatization, wherein words are transformed into their root forms. This not only improves consistency but also aids in further reducing dimensionality while retaining the essential semantic information. The overarching goal of this preprocessing approach is to enhance the quality of the textual data, minimizing noise and ensuring its readiness for subsequent analysis or utilization in machine learning algorithms tailored for software failure prediction

C. EDA

Exploratory Data Analysis (EDA) stands as a pivotal phase in the continuum of data analysis, marked by its systematic exploration, visualization, and comprehension of a dataset. It functions as a fundamental tool, empowering data analysts, researchers, and scientists to unearth pertinent insights, identify patterns, detect anomalies, and formulate hypotheses. EDA serves as a crucial precursor to in-depth studies and informed decision-making. Its significance lies in its dual capability: first, to unveil latent information inherent in the dataset, and second, to provide a framework for conducting thorough and comprehensive research.

By cultivating a more comprehensive insight into the data, EDA plays a pivotal role in guiding subsequent stages of analysis, contributing to the formulation of hypotheses, model selection, and the overall refinement of analytical methodologies..

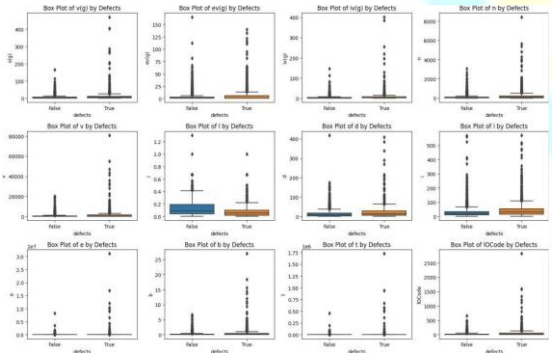

Figure: a box plot depicts the distribution of software defect metrics

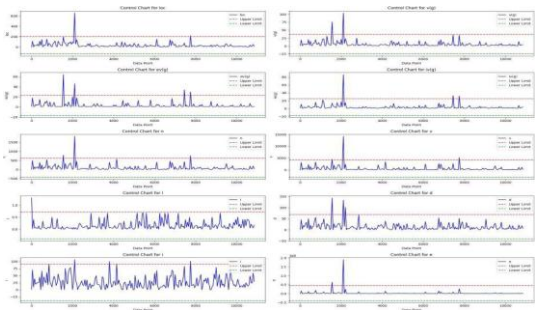

Figure: Informative control charts for the specified Software metrics

D. DATA SPLITTING

In the domain of machine learning for software failure prediction, a widely adopted methodology is the "80:20 data partitioning" strategy, which involves dividing a dataset into training and testing subsets, allocating 80% for training and reserving the remaining 20% for testing. This approach serves as a standard practice due to its effectiveness. By maintaining a clear distinction between training and testing data, this methodology establishes a framework for assessing the credibility of predictive and analytical outcomes. The evaluation process involves contrasting model-generated results with benchmarks derived from the reserved testing subset, ensuring a thorough assessment of the model's capabilities.

E. MODELS USED

The diagnostic approaches for identifying depression. The application of machine learning techniques to the detection of depression involves leveraging specialized algorithms, specifically the Multilayer Perceptron (MLP),,the Decision Tree Classifier, K Neighbors Classifier, Gaussian Naive Bayes (Gaussian NB), and Support Vector Classification (SVC). These algorithms are adept at analyzing various characteristics and patterns to formulate effective

- The Multi-Layer Perceptron (MLP) algorithm is a type of artificial neural network employed in machine learning, specifically for software failure prediction. Comprising an input layer, hidden layers, and an output layer, each layer consists of nodes connected by weighted edges. MLP, with its versatility, is widely used for tasks like system health monitoring and predictive analytics in software reliability, offering a robust framework to discern intricate patterns and enhance predictive accuracy.

- The Decision Tree Classifier is chosen for its ability to discern complex relationships within the data. This characteristic empowers the model to potentially capture nuanced features contributing to depression, enabling a more comprehensive understanding of the underlying factors

- The K Neighbors Classifier, on the other hand, excels in identifying individuals with similar feature patterns. By relying on the similarity of examined cases, it becomes proficient in detecting shared characteristics among individuals experiencing depressive episodes. This capability facilitates a more personalized and precise approach to depression diagnosis.

- The SVC algorithm is employed due to its efficiency in defining hyper planes within multi dimensional feature space. This attribute makes it a potent method for classifying individuals into depressive and non-depressive categories based on detailed feature patterns. The SVC's

ability to create clear boundaries in feature space enhances its accuracy in distinguishing between individuals with and without depression.

- Gaussian Naive Bayes is a classification algorithm in machine learning that uses Bayes' theorem and assumes features are normally distributed. It's particularly useful for handling continuous data and is known for its simplicity and effectiveness. This algorithm assumes that each feature is independent of the others, given the class label. Gaussian Naive Baye's assumes that each parameter, also called features or predictors, has an independent capacity of predicting the output variable.

V. RESULTS

By scrutinizing the models' responses to different algorithms, this analysis aims to elucidate the strengths and limitations of each model, thereby informing the selection and optimization of models for enhanced predictive accuracy in the domain of software failure prediction using Machine Learning.

1) Accuracy

In the domain of classification, accuracy is quantitatively defined as the ratio of correctly classified instances to the total size of the dataset, expressed as a percentage. This fundamental metric serves as a rigorous measure of a classification model's effectiveness in accurately assigning data examples to their respective categories.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

2) Loss :

In instances where anticipated outcomes diverge from actual observations, the consequential outcome is often disappointment. This process aims to optimize the model's performance by mitigating discrepancies between predicted and actual outcomes, thereby enhancing its utility and reliability.

$$Loss = -\frac{1}{m}\sum_{i=1}^{m} fi.\log(fi)$$

3) Precision

Precision is a metric that assesses the model's performance in making accurate predictions, specifically measuring how often the model correctly anticipates a favorable outcome. This statistical measure addresses the question of how many times the model accurately predicts positive instances. Mathematically, precisionis expressed as the ratio of true positives to the sum of true positives and false positives

$$Precision = \frac{TP}{TP+FP}$$

4) Recall

The evaluation of model performance hinges on its ability to recall and correctly identify all pertinent data points. Specifically, when posed with the question, "Among all the genuine positive instances, how many did the model accurately predict as positive?" recall furnishes the relevant answer. In essence, while a high recall indicates the model's proficiency in capturing positive instances, a balanced consideration of other metrics is imperative for a comprehensive assessment of its overall performance.

$$Recall = \frac{TP}{TP+FN}$$

5) F1 - Score

The F1-score serves as a consolidated metric that integrates a classifier's recall and precision through the calculation of their harmonic mean. This metric is specifically designed for the comparative evaluation of two classifiers.

$$F1-score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

Performance Evaluation of Machine Learning Model

Models	Accuracy	Precision	Recall	F1-score
MLP model	0.93	0.70	0.92	0.79
Gaussian NB	0.98	0.92	0.96	0.94
Decision tree	0.99	1.0	1.0	1.0
SVC	0.97	0.97	0.80	0.87
KNN	0.99	0.98	0.96	0.93
Ensemble	0.99	1.0	1.0	1.0

The table provides a thorough assessment of machine learning models for software defect prediction, using important performance measures. The rows in the table represent individual models, while the columns provide information on several metrics, including accuracy, precision, recall, and F1-score.

The MLP model demonstrates a commendable accuracy of 0.93, suggesting that it consistently makes correct predictions.

Gaussian Naive Bayes (NB) demonstrates exceptional performance in defect prediction, with a high accuracyof0.98 and well-balanced precision, recall, and F1-score of 0.92, 0.96, and 0.94 respectively.

The Decision Tree algorithm demonstrates outstanding prediction capabilities, achieving perfect scores (1.0) in accuracy, precision, recall, and F1-score, across all measures.

The Support Vector Classifier (SVC) demonstrates a commendable accuracy of 0.97.

KNN has exceptional performance with a high level of accuracy(0.99) and well-balanced precision (0.98), recall (0.96), and F1-score (0.93).

The Ensemble method, which mirrors the Decision Tree, receives perfect scores (1.0) in all measures, confirming its effectiveness in predicting software defects.

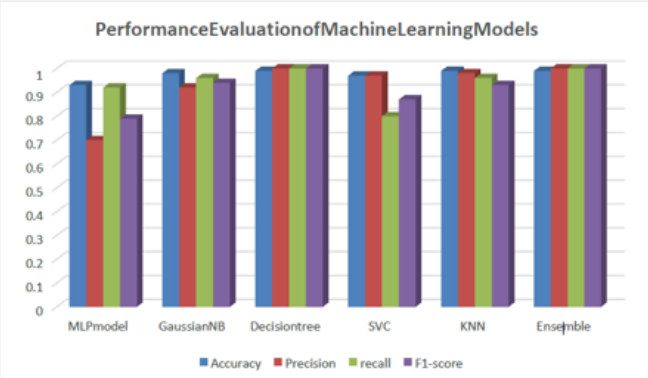

Figure: Performance evaluation graph of machine learning models

The Decision Tree and Ensemble models demonstrate exceptional predictive capabilities with perfect scores across all metrics, making them suitable for tasks where precision, recall ,and overall accuracy are crucial. Gaussian NB excels with a balanced approach, showcasing high accuracy and well-maintained precision, recall, and F1-score, making it a reliable choice for defect prediction. KNN, with its high accuracy and balanced precision and recall, offers a robust solution for identifying software defects.

VI. SUMMARY

Software defects can have a severe impact on software quality, causing problems for customers and developers. With growing complexities in software designs and technology, manual software detection becomes a challenging and time-consuming task. Thus, automatic software detection has become a hotspot for industrial research in the past couple of years. In this paper, we try to apply machine learning and deep learning to solve this problem. We use datasets provided by the NASA Promise dataset repository and compare the state of the art machine learning algorithms' results. The strengths and weaknesses of each model are highlighted in these measures, helping choose the best model depending on specific goals and trade-offs between precision and recall in software defect prediction tasks. This field still has much scope for improvement. We can think of some novel approaches which use complex deep learning algorithms, and also researchers should focus on more data collection

REFERENCES

[1]. Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv –
[2]. Y.Cai,Softwarereliabilityengineeringfoundation,Tsinghua universitypress,1995.
[3]. J.Gao,L.Zhang,Z.FengrongandZ.Ye,"ResearchonSoftwareClassification,"inInformationTechnology,Networking, Electronic and Automation ControlConference,2019.
[4]. I.C.Society,"IEEE729-1983 IEEEStandardGlossaryofSoftware Engineering Terminology," 1982.
[5]. W.Bi,"ResearchonSoftwareDefectClassificationandAnaly sis,"ComputerScience,2013.
[6]. YangandM.Duan,"ResearchofSoftwareDefectAnalysisTechnology,"Computer Engineering &Software,2018.
[7]. J.CollofelloandB.P.Gosalla,"Anapplicationofcausalanalysistothesoftwaremodificationprocess,"Software:Practiceand Experience, vol. 23, 1993.
[8]. J.W.Horch, Practica l Guide to Software Quality Management Artech House,2003.
[9]. R. Chillarege, I. Bhandari, J. Chaar, M. J. Halliday, D.S.Moebus, B.K.RayandM .-Y.Wong, "Orthogonal Defect Classification- A Concept for In-Process Measurements,"IEEETransactionsonsoftwareEngineering,vol.18,pp.943-956,1992.
[10]. S.&.S.E.S.Committee,"IEEE1044-1993-IEEE StandardClassificationforSoftwareAnomalies,"IEEE,1993.
[11]. X.Huang,Softwarereliability,safetyandqualityassurance,Electronic Industry Press, 2002.
[12]. L. Putnam and W. Myers, Measures forexcellence:reliablesoftwareontime,withinbudget,PrenticeHallProfessional Technical Reference, 1991.

[13]. I. Raphael and C. Michael, "Fault links:identifyingmoduleandfaulttypesandtheirrelationship,"2004.
[14]. L.Macaulay,Humanomputerinteractionforsoftwaredesigners,Itp-Media,1995.
[15]. "NASAPromiseDatasetRepository". I. T. Jolliffe and J. Cadima, "Principalcomponentanalysis: a review and recentdevelopments,"Philosophical Transactions of theoyalSocietyA:Mathematical,PhysicalandEngineering Sciences,vol.374,2016.
[16]. L.Breiman,"RandomForests,"MachineLearning,vol.45,pp.5-32, 2004.
[17]. Y.LeCun,Y.BengioandG.Hinton,"Deeplearning,"Nature,vol.521,pp.436-444,2015.
[18]. "Logisticregression,"Wikipedia.
[19]. "NaiveBayes,"Wikipedia.
[20]. "GradientBoostingClassifier,"Wikipedia.
[21]. "SupportVectorMachine,"Wikipedia.
[22]. P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta,T.Duan,D.Ding,A.Bagul,C.Langlotz,K.Shpanskaya,M.Lungren and A. Ng, "CheXNet: Radiologist-LevelPneumonia Detection on Chest X-Rays withDeepLearning,"Arxiv, vol. abs/1711.05225, 2017.
[23]. J.BaandD.P.Kingma,"Adam:AMethodforStochastic Optimization,"Clinical Orthopaedics and RelatedResearch,