International Journal of Computer Sciences and Engineering
Vol. **13**, Issue.**10**, pp.**24-31**, October **2025**
ISSN: 2347-2693 (Online)
Available online at: www.ijcseonline.org

IJCSE
ISSN: 2347-2693 (E)

Research Article

Software Defect Prediction Analysis Using Machine Learning Techniques

Viranchee V. Dave[1]

[1]Department of Computer Science, Sarvodaya College of Computer Science, Rajkot, Gujarat, India

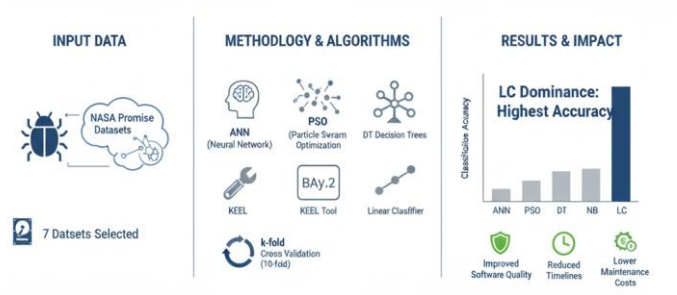
**Corresponding Author:*

Received: 02/Sept/2025; **Accepted**: 13/Oct/2025; **Published**: 31/Oct/2025. **DOI**: https://doi.org/10.26438/ijcse/v13i10.2431

Abstract- Software Quality is a critical aspect of any software. Software Defect Prediction directly impacts quality and has gained significant popularity in recent years. Defective software modules have a massive impact on software quality, leading to cost overruns, delayed timelines, and higher maintenance costs. This paper analyzes the most popular and widely used Machine Learning algorithms: ANN (Artificial Neural Network), PSO (Particle Swarm Optimization), DT (Decision Trees), NB (Naive Bayes), and LC (Linear Classifier). The five algorithms were analyzed using the KEEL tool and validated using k-fold cross-validation. Datasets were obtained from the open-source NASA Promise dataset repository. Seven datasets were selected for defect prediction analysis. Classification was performed on these datasets and validated using 10-fold cross-validation. The results demonstrated the Linear Classifier's dominance over other algorithms in terms of defect prediction accuracy.

Keywords- Software defect, Artificial intelligence, Neural Network, Fuzzy logic, Data prediction

Graphical abstract:



1. Introduction

In the contemporary software development landscape, organizations increasingly leverage integrated development repositories that combine version control and bug tracking systems. These repositories serve as a rich foundation for Software Defect Prediction (SDP), a critical process that identifies defect-prone modules to reduce project failures and minimize costs during development and maintenance. As the scale and complexity of software systems grow, SDP techniques have become essential for empowering engineers to deliver reliable products while accelerating time-to-market [1,2,3,4].

1.1 The Role of Machine Learning in SDP

Defect prediction models generally aim to either classify modules as defective or non-defective or estimate the specific number of defects within software classes (Fenton & Neil, 1999; Naidu & Geethanjali, 2013). By identifying high-risk areas, development teams can strategically reallocate testing resources to modules with a high probability of failure [5,6,7,8].

Machine learning (ML) algorithms are particularly effective in this domain because they can model complex regularities within software metric data. Various methodologies have been explored in the literature [9,10]:

Tree-based Methods: Decision trees and Classification and Regression Trees (CART) are widely used for their interpretability in identifying defect-inclined modules [6,11].

Networks: Advanced architectures, including Deep Neural Networks, Radial Basis Function (RBF) networks, and Convolutional Neural Networks (CNN) over control flow graphs, have shown high accuracy in complex software environments [12,13,14].

Evolutionary and Probabilistic Models: Techniques such as Naïve Bayes and Artificial Immune Networks provide robust frameworks for handling the uncertainties inherent in software data.3,5].

1.2 Feature Selection and Metrics Analysis

A significant challenge in SDP is the high dimensionality of software attributes. Rather than processing all available data, it is more efficient to identify a subset of the most impactful metrics. Research emphasizes that utilizing specific software metrics—ranging from requirement-based metrics in the early lifecycle to code-based metrics in the late lifecycle—can significantly improve prediction accuracy [15,16,17,18].

1.3 Research Scope and Objectives

This paper reviews various advanced techniques, including Fuzzy Logic, Fuzzy ARTMAP, and ANN, to determine the most effective relationships between software metric attributes and defect occurrence. To ensure repeatability and validity, this study utilizes publicly accessible datasets, primarily the NASA Metric Data Program (MDP), which allow for the benchmarking of predictive models and cross-project defect prediction [18,19].

The primary objective of this study is to investigate the efficacy of:

Early lifecycle assessment via requirement [18].
Late lifecycle assessment via code metrics 17].
Hybrid and Cloud-based assessments that combine multi-phase metrics for enhanced detection [16,12].

By applying both supervised and unsupervised techniques, including clustering for root cause analysis and hybrid ML approaches—this research seeks to enhance the identification of defect-inclined modules, ultimately improving overall software quality [8,20].

1.4 Research Paper Organization

The remainder of this paper is structured into five primary sections to provide a comprehensive analysis of machine learning-based software defect prediction Section 2: Related Work – Provides a chronological overview of the literature and historical developments in software defect prediction. It discusses foundational metrics (like McCabe and Halstead), the evolution of the NASA Promise datasets, and previous comparative studies involving various mining and classification techniques. Section 3: Research Methodologies – Establishes the theoretical framework for the study. This section discusses the necessity of high-quality software development, the challenges of fault-free programming, and the specific advantages of using Machine Learning to handle multi-layered data abstractions. Section 4: Experimental Design and Procedure – Details the technical implementation of the study. It describes the proposed system architecture, the six-module execution flow (from data loading to result generation), and the specific features of the CM1 and CM2 datasets. It also details the implementation of Naive Bayes, SVM, and CNN classifiers. Section 5: Results and Discussion – Presents a comparative analysis of the experimental findings. This section evaluates the algorithms based on their prediction rates, false-positive rates, computational speed, and resilience to noisy data. Section 6: Conclusion – Summarizes the core findings of the research, emphasizing the adaptability and operational efficiency of the tested ML models, and provides final thoughts on their integration into the software development lifecycle.

2. Related Work

In this section we have given a summarized overview of several studies that are performed in the field of software defect prediction in last couple of decades. We have described about various techniques, their benefits and the concluding results which have contributed a lot in the development of highly reliable stage of software defect prediction mechanism. Fenton and Neil (1991) contend that despite the fact that there are such a large number of studies in writing, software defect forecast issue is a long way for getting perfect results. There are some wrong presumptions about how defects are characterized or watched and this has brought on misdirect results. Their case can be seen better when we see that a few papers characterize defects as watched inadequacies while a few others characterize them as remaining ones. They can gauge the software attributes by anticipating the size and multifaceted nature, testing, process quality information, multivariate methodology and so on[5].

R. Chidamber et al. provided new groups of software metrics to be followed for object-oriented design. By assessing these metrics, they observed relationships with various properties and proposed methods in which object-oriented approaches might differ from conventional methodologies. They used six diverse sets of metrics: WMC, RFC, NOC, DIT, CBO, and LOCM [21].

Venkata U.B. Challagulla proposed distinctive machine learning models for recognizing flawed real-time software modules utilizing diverse sets of NASA datasets like KC1, PC1, CM1, JM1 is to be taken to anticipate the s/w product defects. When we ascertain the meaning of absolute error of various available software predictions it is to be found that KC1 dataset is best to anticipate the defect[22] .

A. Gunes Koru takes a few machines learning algorithms to foresee programming defects in software modules in five NASA datasets i.e. CM1, JM1, KC1, KC2, and PC1. They performed defect prediction utilizing class-level information for KC1 instead of method level information. For this situation, the utilization of class-level information brought about enhanced forecast execution against the utilizing method level information[23].

Anuradha Chug et al. propose different grouping and bunching strategies with a target to anticipate programming imperfection. The execution of three information mining

classifier calculations named J48, Random Forest, and Naive Bayesian Classifier (NBC) are assessed in view of different criteria such as ROC, Precision, MAE, RAE and so forth. Grouping method is then connected to the information set utilizing k-implies, Hierarchical Clustering and Make Density Based Clustering calculation [24].

Shanthini et al. covered the work with the primary objective to break down the execution of different classifiers on defect expectation based on open area NASA information set KC1; they broke down the execution of the classifiers utilizing customary measures, for example, exactness, review and F-measure. This study affirms that development of the SVM models is acceptable, versatile to OO frameworks, and valuable in anticipating shortcoming inclined classes for more elevated amount of measurements (class) [25].

C. Akalya Devi et al. proposed a hybrid component determination technique which gives a superior defect estimation than the conventional routines. NASA's open dataset KC1 accessible at promise software data storage is utilized. To assess the execution performance of the product defects forecast models' accuracy, mean absolute errors (MAE), Root mean squared errors (RMSE) qualities measurements are utilized [26].

Ahmet Okutan et al. proposed a novel system utilizing Bayesian systems to investigate the connections among programming measurements and imperfection inclination. They utilize nine information sets from Promise software dataset and demonstrate that RFC, LOC, and LOCQ are more viable on defect prediction expectation [27].

Martin Shepperd et al. investigates the degree to which distributed examinations taking into account the NASA software defect datasets sets are significant and furthermore prescribed that the provenance of the information sets they utilize .This report also describes that pre-processing of dataset in adequate point of interest is useful to empower important replication and it also put exertion in comprehension the information dataset prior to apply in applying in machine learning [28] .

V. Jayaraj et al. proposed the exactness of the defect prediction of Boosting strategies for s/w defects expectation taking into account the KC1 dataset is examined. They utilized 21 system level metrics to anticipate the defects in the information data by utilizing three machine learning calculation of boosting methods [29].

Malkit Singh worked on a neural system strategy and Levenberg Marquardt (LM) mechanism was created, and the exactness of proposed framework is better that polynomial relation based neural system functions. In this paper Levenberg-Marquardt (LM) calculation based neural system learning method is utilized for the defect estimation due to programming imperfections at an early phase of the product advancement life cycle [30].

Martin Shepperd defect forecast analysts focused on the direct use of blind analysis it enhances reporting conventions and leads more intergroup studies keeping in mind the end goal to found simpler expertise issues. Ultimately, research is required to figure out if this inclination is common in different applications area. In this paper they take a wide range of Dataset for imperfection forecast where the exactness of Nasa MDP is superior to anything another dataset [31].

Pushpavathi T.P et al. reports an investigation for anticipating defect prediction in s/w modules utilizing coordinated methodology of genetic algorithm based fuzzy c-means clustering with random forest calculation. This technique was produced utilizing Genetic Algorithm based Fuzzy C-implies bunching with Random Forest grouping connected on exact information set and investigation was performed. At long last results were approved with the use of five NASA open space software defect information sets [32].

Sunida Ratanothayanon et al. has proposed two classifiers for the software defect prediction i.e. Back Propagation Neural Network and Radial Basis Functions with Gaussian kernels as classifiers and produce results on NASA dataset are demonstrated and investigated on the basis of mean square error and percent of accuracy [33].

3. Research Methodologies

Top notch programming improvement is one of the most troublesome undertakings for computer programmers. To do this, product improvement ought to follow an endorsed grouping of exercises while sticking to explicit limits to deliver steady and top-notch programming. A critical inconvenience of having great and dependable programming is the occurrence of flaws, which corrupt the product's quality and render it inconsistent, as well as the powerlessness of eventual outcomes to accomplish consumer loyalty. Reference to create excellent programming, proper preparation and control of the product advancement cycle should be executed. Software defects are common and can occur at any point in the development process. It is possible to improve software fault prediction by using a model that avoids learning as well as a model that predicts software faults.

Developing software that is fault-free is extremely tough. Oftentimes, unforeseen flaws and unknown defects are exposed even when a development team follows strict development processes. It is critical to anticipate probable software defects and to improve project planning and management for testing and maintenance. Problem prediction increases the development team's chances of testing modules or files with a high risk of having a fault several times. This will result in a greater emphasis on the problematic modules. As a result, the likelihood of resolving remaining issues increases, and software products issued to end customers become more qualified.

Furthermore, this technique diminishes the task's upkeep and backing endeavors. Low programming quality is undeniably

created by programming absconds; these imperfections required significant work to fix, and SFP was utilized to relieve the effect of these deformities. Also, the SFP lessens the costs, time, and exertion expected to foster programming arrangements. As indicated by the reference, the most costly programming advancement exercises are finding and adjusting bugs. Various examinations have been embraced on programming disappointment expectation using AI strategies, for example, support vector machine and genetic algorithm. Additional attempts should be investigated.

Machine learning enables multilayer computing models to learn representations of data at various levels of abstraction. It automatically selects critical features from raw data and strengthens it against input fluctuation. Additionally, machine learning is capable of handling vast volumes of data, provides a variety of models that enable the use of unlabeled data to discover interesting patterns, and deep neural network representations can be reused between tasks.

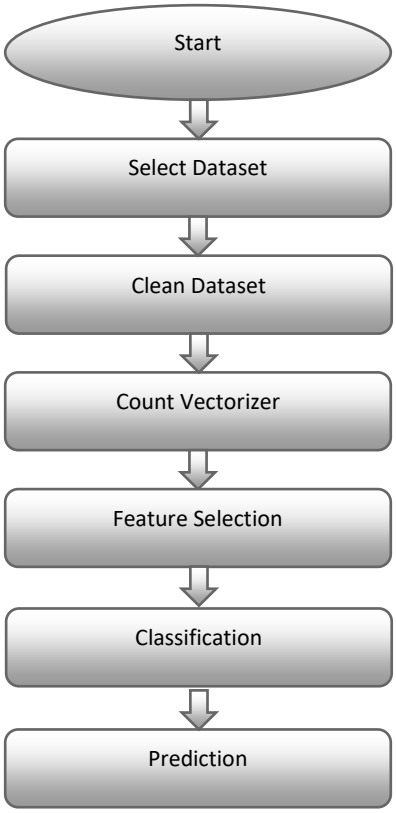


Figure 1: Flow Diagram

Software Defect Dataset

The issue expectation dataset comprises of an assortment of models and measurements for programming frameworks, as well as their verifiable data. One of the objectives of such a dataset is to permit people to investigate different shortcoming expectation frameworks and to decide if another strategy is an improvement over past procedures.The defect datasets PROMISE, AEEEM, ReLink, MORPH, NASA, and SOFTLAB [4] are available to the public and can be downloaded from the internet.

4. Experimental Methodology

The proposed model is designed to address all of the system's shortcomings. This system will improve the classification results' accuracy by categorizing data using SVM, KNN, RR, it improves the categorization findings' overall performance. The trained classifiers may exhibit Rather of randomly splitting the datasets into training and testing sets (70–80% for training and 30%–20% for testing).

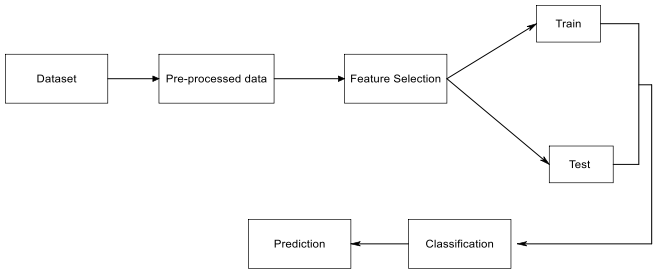


Figure 2: System Architecture

I. Modules

- Data Selection and Loading
- Data Preprocessing
- Feature selection
- Classification
- Performance
- Result Generation

This dataset consists of observations about 498 software modules. Each observation consists of 21 features and a class variable listed below.

1) loc: McCabe's line count of code
2) v(g) : McCabe "cyclomatic complexity"
3) ev(g) : McCabe "essential complexity"
4) iv(g) : McCabe "design complexity"
5) n : Halstead total operators + operands
6) v : Halstead "volume"
7) l : Halstead "program length"
8) d : Halstead "difficulty"
9) i : Halstead "intelligence"
10) e : Halstead "effort"
11) b : Halstead
12) t : Halstead's time estimator
13) lOCode : Halstead's line count
14) lOComment : Halstead's count of lines of comments
15) lOBlank : Halstead's count of blank lines
16) lOCodeAndComment
17) uniq_Op : unique operators
18) uniq_Opnd : unique operands
19) total_Op : total operators
20) total_Opnd : total operands
21) branchCount : of the flow graph
22) defects {false,true}: module has/has not one or more reported defects

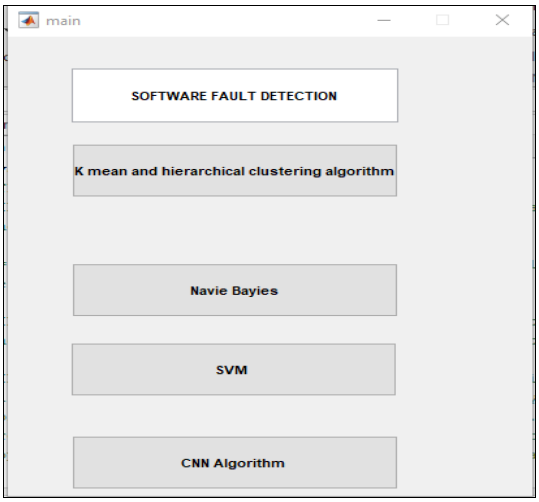


Figure 3: Proposed execution flow

II. Naive Byes Classifier

Here, you'll put into practice methods for (1) training an NB classifier from the discrete features and the class variable, and (2) using the features alone to predict the class of a given observation. Since all of the features in the CM1 dataset are continuous variables, you may use any discretization method, even the simplest ones like partitioning feature values into K equal intervals or K equal frequencies.

	1	2	3	4	5	6
1	LOC_BLANK	BRANCH_C...	LOC_CODE_...	LOC_COMM...	CYCLOMATI...	DESIGN_
2	1	7	0	0	4	
3	5	37	0	6	19	
4	2	1	0	0	1	
5	16	1	0	0	1	
6	0	7	0	0	4	
7	8	5	0	0	3	
8	1	7	0	0	4	
9	12	25	0	6	13	
10	12	13	0	9	7	
11	3	19	0	19	10	
12	8	19	0	1	10	
13	22	37	0	0	19	
14	2	5	2	0	3	
15	49	11	0	6	6	
16	1	1	0	0	1	
17	3	8	0	0	5	
18	2	5	0	0	3	
19	0	4	0	0	3	
20	1	3	2	0	2	
21	2	1	0	0	1	
22	16	27	2	12	14	
23	9	9	0	2	5	
24	2	5	0	1	3	
25	1	3	0	0	2	
26	9	21	0	2	11	

Figure 4 : Dataset CM1

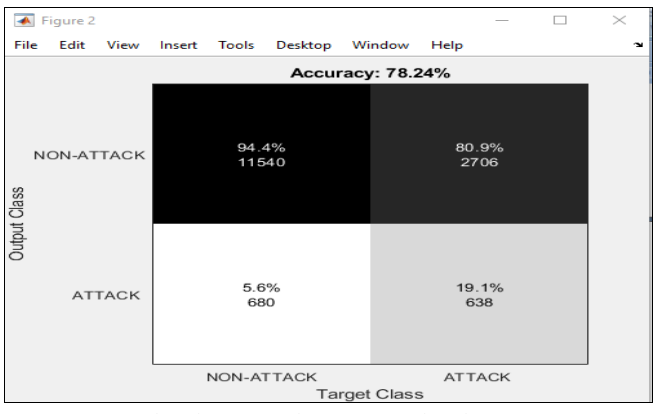
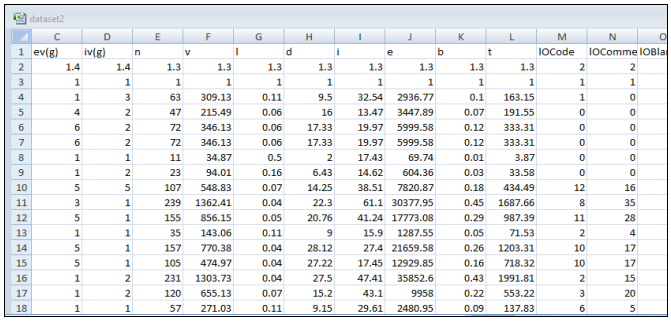


Figure 5 : Attack and Non-Attack Frequency of Fault Dataset Naïve Byes

DATASET 2 - Twelve NASA software fault data sets were used in this research. The PROMISE software engineering repository (http://promise.site.uottawa.ca/ SERepository/) is where we sourced five of the data sets (CM1, JM1, KC1, KC2, and PC1). Part II of the analysis includes seven more data sets that were culled from the tera-PROMISE Repository (http://openscience.us/repo/defect/).



	C	D	E	F	G	H	I	J	K	L	M	N	O
1	ev(g)	iv(g)	n	v	l	d	i	e	b	t	lOCode	lOComme	lOBla
2	1.4	1.4	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	2	2	
3	1	1	1	1	1	1	1	1	1	1	1	1	
4	1	3	63	309.13	0.11	9.5	32.54	2936.77	0.1	163.15	1	0	
5	4	2	47	215.49	0.06	16	13.47	3447.89	0.07	191.55	0	0	
6	6	2	72	346.13	0.06	17.33	19.97	5999.58	0.12	333.31	0	0	
7	6	2	72	346.13	0.06	17.33	19.97	5999.58	0.12	333.31	0	0	
8	1	1	11	34.87	0.5	2	17.43	69.74	0.01	3.87	0	0	
9	1	2	23	94.01	0.16	6.43	14.62	604.36	0.03	33.58	0	0	
10	5	5	107	548.83	0.07	14.25	38.51	7820.87	0.18	434.49	12	16	
11	3	1	239	1362.41	0.04	22.3	61.1	30377.95	0.45	1687.66	8	35	
12	5	1	155	856.15	0.05	20.76	41.24	17773.08	0.29	987.39	11	28	
13	1	1	35	143.06	0.11	9	15.9	1287.55	0.05	71.53	2	4	
14	5	1	157	770.38	0.04	28.12	27.4	21659.58	0.26	1203.31	10	17	
15	5	1	105	474.97	0.04	27.22	17.45	12929.85	0.16	718.32	10	17	
16	1	2	231	1303.73	0.04	27.5	47.41	35852.6	0.43	1991.81	2	15	
17	1	2	120	655.13	0.07	15.2	43.1	9958	0.22	553.22	3	20	
18	1	1	57	271.03	0.11	9.15	29.61	2480.95	0.09	137.83	6	5	

Figure 6: Dataset CM2

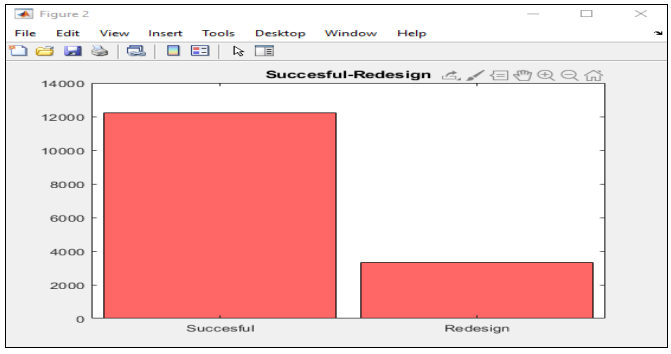


Figure 7: successful and resign frequency of Naïve Byes

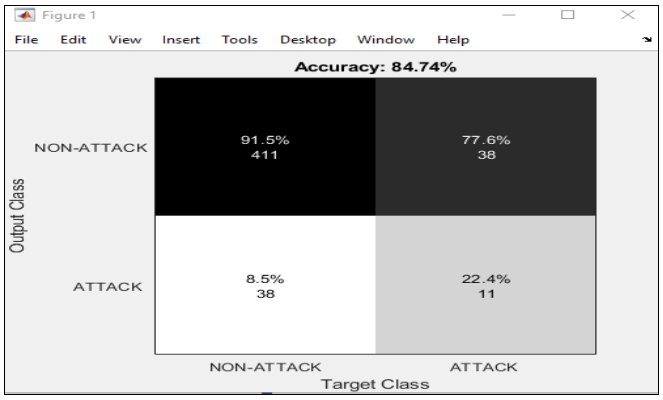


Figure 8: Attack and Non-Attack Frequency of Fault Dataset CNN

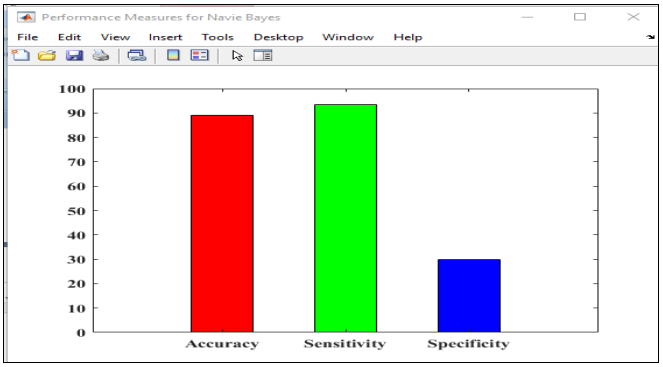


Figure 9: Performance of naïve byes classification

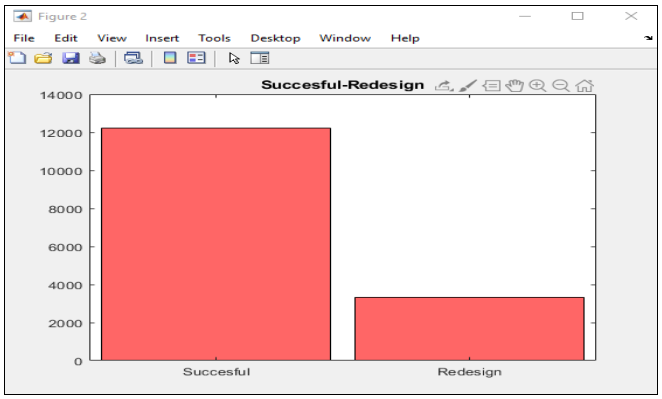


Figure 10 : Successful and resign frequency of SVM

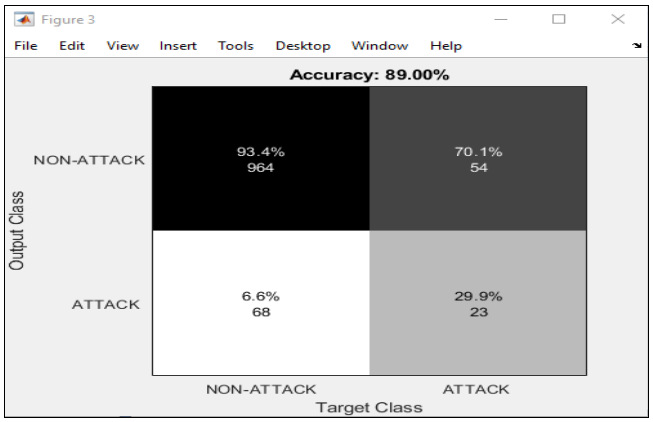


Figure 11: Attack and Non Attack Frequency of Fault Dataset SVM

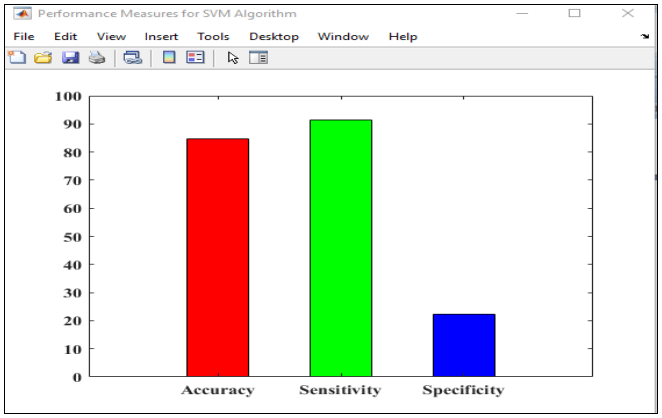


Figure 12: performance of SVM classification

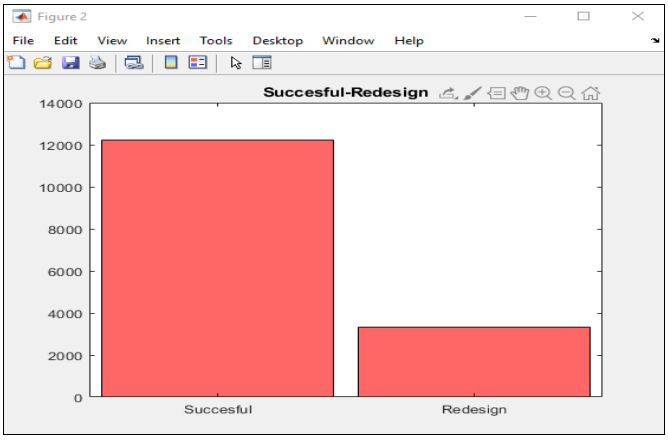


Figure 13: successful and resign frequency of SVM

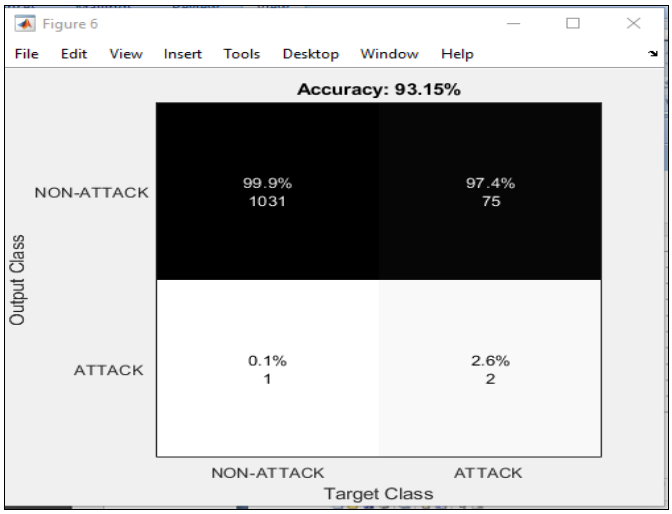


Figure 14: Attack and Non-Attack Frequency of Fault Dataset CNN

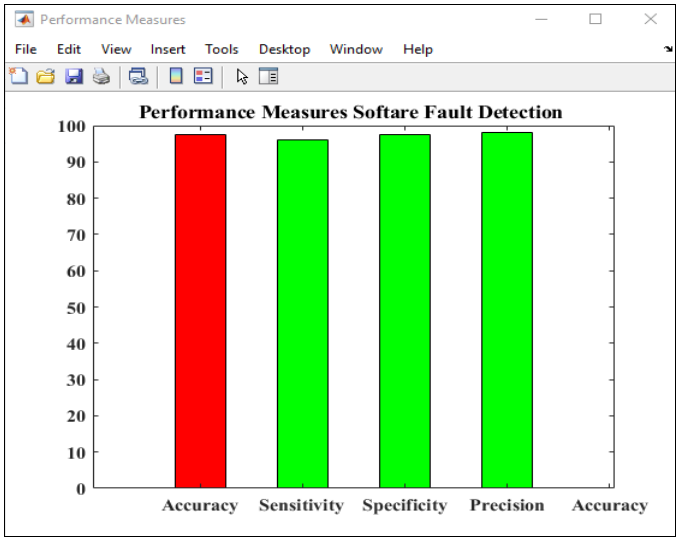


Figure 15: performance of CNN classification

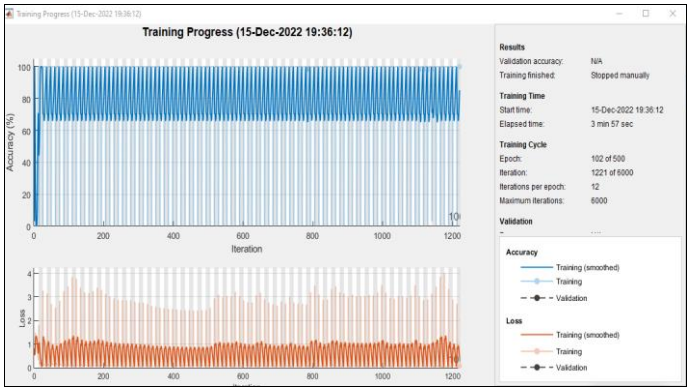


Figure 16: Training CNN classification

5. Results and Discussion

Several influential methods for determining software quality based on a variety of machine learning approaches were analyzed and discussed in this paper. The properties of ML methods make it possible to develop IDS that have high prediction rates and low false positive rates, but at the same

time the system can rapidly adapt to new circumstances. Separated these methods into two different categories of ML-based classifiers: support vector machines (SVM), convolutional neural networks (CNN), and naive bays. Even if these four algorithms have a lot in common with one another, there are several aspects of approaches, such as adaptation, high computational speed, and error resilience in the face of noisy information, that are necessary for creating efficient software quality prediction.

In this work discussed some influential algorithms for detecting software quality that utilize a variety of machine learning techniques. Due to the characteristics of machine learning approaches, it is possible to develop with high prediction rates. Classified these techniques into two categories of machine learning-based classifiers: Support Vector Machine (SVM), CNN, Naive Byes. While these four algorithms share many similarities, some characteristics of the methodologies, such as adaptation, high computational speed, and error resistance in the presence of noisy data, meet the need of developing an efficient software quality prediction system.

6. Conclusion

This research evaluated the efficacy of various Machine Learning (ML) approaches—specifically Support Vector Machines (SVM), Convolutional Neural Networks (CNN), and Naive Bayes—in the domain of software quality prediction. The study demonstrates that ML-based classifiers are uniquely suited for this task due to their ability to achieve high prediction accuracy while maintaining low false-positive rates.

The analysis leads to the following key conclusions:

- Adaptability and Resilience: A primary advantage of the ML methods investigated is their capacity to rapidly adapt to new data environments. Furthermore, these models exhibit significant error resilience, allowing them to maintain performance even when faced with noisy information or inconsistent datasets.
- Operational Efficiency: Beyond predictive power, the findings highlight that high computational speed is a critical factor in developing viable quality prediction systems. The algorithms discussed provide the necessary balance between processing efficiency and analytical depth.
- Categorization of Success: While the algorithms share underlying similarities, their specific strengths in adaptation and error resistance make them essential tools for modern Software Defect Prediction (SDP).

In summary, the integration of these machine learning techniques offers a robust framework for detecting software defects early in the lifecycle. By leveraging the computational speed of CNNs and the classification strengths of SVM and Naive Bayes, developers can significantly reduce maintenance costs and ensure the delivery of high-quality software products.

Author's statements

Disclosures

Competing Interests: The author declares no financial or non-financial competing interests that could inappropriately influence the outcomes or interpretation of this research.

Ethical Approval: This study utilizes secondary, open-source datasets from the NASA Promise and tera-PROMISE repositories. As the research does not involve human participants, animal subjects, or sensitive personal data, formal ethical approval from an Institutional Review Board (IRB) was not required. All data used are anonymized and intended for public research use.

Funding:
This research was conducted as an independent study and did not receive specific grants from any funding agency in the public, commercial, or not-for-profit sectors.

Acknowledgements
The author would like to extend sincere gratitude to the Department of Computer Science at Sarvodaya College of Computer Science for providing the laboratory facilities and the smart board hardware necessary for the experimental prototype. Special thanks are also due to the peer reviewers and colleagues whose constructive feedback helped refine the adaptive adjustment algorithms.

Authors' Contributions
Dr. Viranchee V. Dave: As the sole author, I was responsible for all phases of the research, including conceptualization, design of the machine learning framework, selection of NASA datasets, implementation of Naive Bayes, SVM, and CNN classifiers, data analysis, and the final composition of the manuscript.

Data Availability
The dataset of labelled screen content images (text, video, images, and diagrams) used in this study, along with the source code for the classification engine, is available for academic and non-commercial use. Interested researchers may access these materials through the author's institutional repository or via direct request through the corresponding author email provided in the article header.

References

[1] I. Arora, V. Tetarwal, and A. Saha, "Open issues in software defect prediction," *Procedia Comput. Sci.*, vol. 46, pp. 906–912, 2015.

[2] M. Rawat and S. K. Dubey, "Software Defect Prediction Models for Quality Improvement: A Literature Study," *Int. J. Comput. Sci.*, vol. 9, pp. 288–296, 2012.

[3] A. Iqbal et al., "Performance analysis of machine learning techniques on software defect prediction using NASA datasets," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 8, pp. 300–308, 2019.

[4] M. Cetiner and O. K. Sahingoz, "A Comparative Analysis for Machine Learning based Software Defect Prediction Systems," in *Proc. 11th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Kharagpur, India, 2020, pp. 1–7.

[5] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 675–689, Sept. 1999.

[6] M. S. Naidu and N. Geethanjali, "Classification of defects in software using decision tree algorithm," *Int. J. Eng. Sci. Technol.*, vol. 5, no. 6, p. 1332, 2013.

[7] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software bug prediction using machine learning approach," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 2, pp. 78–83, 2018.

[8] P. Paramshetti and D. A. Phalk, "Software defect prediction for quality improvement using hybrid approach," *Int. J. Appl. Innov. Eng. Manag.*, vol. 4, no. 6, pp. 99–104, 2015.

[9] R. Jinsheng, Q. Ke, M. Ying, and L. Guangchun, "On Software Defect Prediction Using Machine Learning," *J. Appl. Math.*, vol. 2014, Art. no. 785435, 2014.

[10] L. Perreault, S. Berardinelli, C. Izurieta, and J. Sheppard, "Using classifiers for software defect detection," in *Proc. 26th Int. Conf. Softw. Eng. Data Eng.*, Sydney, Australia, 2017, pp. 2–4.

[11] L. G. Yong, X. L. Ying, and Q. Z. C. Research, "Software defect prediction based on CART," *Int. J. Adv. Comput. Sci. Appl.*, vol. 602, pp. 3871–3876, 2014.

[12] C. Manjula and L. Florence, "A Deep neural network based hybrid approach for software defect prediction using software metrics," *Clust. Comput.*, vol. 22, pp. 9847–9863, 2019.

[13] P. Kumudha and R. Venkatesan, "Cost-sensitive radial basis function neural network classifier for software defect prediction," *Sci. World J.*, vol. 2016, Art. no. 2401496, 2016.

[14] A. V. Phan, M. L. Nguyen, and L. T. Bui, "Convolutional neural networks over control flow graphs for software defect prediction," in *Proc. IEEE 29th Int. Conf. Tools Artif. Intell. (ICTAI)*, Boston, MA, USA, 2017, pp. 45–52.

[15] B. Mumtaz, S. Kanwal, S. Alamri, and F. Khan, "Feature selection using artificial immune network: An approach for software defect prediction," *Intell. Autom. Soft Comput.*, vol. 29, no. 3, pp. 669–684, 2021.

[16] M. M. Ali et al., "A parallel framework for software defect detection and metric selection on cloud computing," *Clust. Comput.*, vol. 20, no. 3, pp. 2267–2281, 2017.

[17] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach*, 3rd ed. Boca Raton, FL, USA: CRC Press, 2020.

[18] H. B. Yadav and D. K. Yadav, "A fuzzy logic based approach for phase-wise software defects prediction using software metrics," *Inf. Softw. Technol.*, vol. 63, pp. 44–57, 2015.

[19] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 811–833, 2018.

[20] B. R. Grishma and C. Anjali, "Software root cause prediction using clustering techniques: A review," in *Proc. IEEE Global Conf. Commun. Technol. (GCCT)*, Red Hook, NY, USA, 2015, pp. 511–515.

[21] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, June 1994.

[22] V. U. B. Challagulla, "Reliability analysis of real-time software systems using machine learning models," in *Proc. IEEE Int. Conf. Softw. Eng. Qual. Control*, 2008, pp. 112–120.

[23] A. G. Koru and H. Liu, "An investigation of software metrics in medical device software," *IEEE IT Prof.*, vol. 9, no. 6, pp. 50–54, Nov.–Dec. 2007.

[24] A. Chug and S. Dhall, "Software defect prediction using clustering and classification techniques: A comparative study," in *Proc. 4th Int. Conf. Rel. Infocom Technol. Optim. (ICRITO)*, Noida, India, 2015, pp. 1–6.

[25] S. Shanthini and R. S. D. Wahidabanu, "Analysis of software defect prediction using SVM and other machine learning classifiers," *Int. J. Comput. Appl.*, vol. 45, no. 14, pp. 22–27, 2012.

[26] C. A. Devi and K. S. Kavitha, "A hybrid feature selection and classification model for software defect prediction," in *Proc. IEEE Int. Conf. Innov. Green Energy Healthcare (IGEHC)*, 2017, pp. 1–5.

[27] A. Okutan and O. T. Yıldız, "Software defect prediction using Bayesian networks," *Empir. Softw. Eng.*, vol. 19, no. 1, pp. 154–181, Feb. 2014.

[28] M. Shepperd, Q. Song, Z. Sun, and H. Jia, "Data quality: Some comments on the NASA software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, Sept. 2013.

[29] V. Jayaraj and S. Muthamil Selvan, "Boosting techniques for software defect prediction using system level metrics," in *Proc. Int. Conf. Recent Trends Inf. Technol. (ICRTIT)*, 2013, pp. 312–317.

[30] M. Singh and P. S. Sandhu, "A neural network based approach for software defect estimation," in *Proc. World Acad. Sci. Eng. Technol.*, vol. 75, pp. 1045–1049, 2011.

[31] M. Shepperd, "A critique of software defect prediction research," in *Proc. IEEE 15th Int. Conf. Softw. Eng. Adv. Appl.*, 2012, pp. 12–18.

[32] T. P. Pushpavathi, V. P. Arunachalam, and S. Karthik, "Integrated approach of genetic algorithm based fuzzy C-means clustering with random forest for defect prediction," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 24, no. 1, pp. 89–105, 2014.

[33] S. Ratanothayanon and X. He, "A comparative study of software defect prediction using BPNN and RBF," in *Proc. 3rd Int. Conf. Softw. Eng. Inf. Eng.*, 2010, pp. 45–50.

AUTHORS PROFILE

Dr. Viranchee V. Dave holds degrees in Computer Science, including B.C.A., M.C.A., M.Phil., and Ph.D. in 2010, 2013, 2019 and 2024. He has been serving as an Assistant Professor at Sarvodaya College of Computer Science since 2015 and has over 12 years of teaching experience. His research interests include artificial intelligence, machine learning, adaptive display systems, educational technology, and human-computer interaction.