Revolutionary Advances in Computing and Electronics (RACE): An International Journal
Volume 02, Issue 02, pp.1-16, 2026
ISSN (Online): 3107-8540

Received: 09/05/2026, Accepted: 01/06/2026

Intelligent Software Fault Prediction Using Machine Learning and Neural Network Techniques

Pooja Singh[1], Saoud Sarwar[2], Kaveri Umesh Kadam[3]

[1,2]Department of Computer Science, Al-Falah University, Faridabad, Haryana

[3]Department of Computer Science, Jamia Hamdard University, New Delhi

research5960@gmail.com

Abstract

The investigative methodology includes a careful process that begins with data cleaning and feature selection. Uses 10-fold cross-validation, which means it basically takes 10ths of the data and tests it to see if it's consistent. A number of machine learning algorithms are evaluated, including Decision Tree, Naive Bayes, Support Vector Machine, Random Forest, and Multi-Layer Perceptron. The classic criteria accuracy, precision, recall and F1 score are used to evaluate all. When the models were completed, the results showed that models that use several techniques (e.g., Random Forest) and/or neural networks (e.g., Multi-Layer Perceptron) generally outperform simpler models. Among them, the Multi-Layer Perceptron is best suited for predictions, and the Random Forest is strong across all types of datasets. These discoveries indicate that machine learning can be used to estimate software failure time. Based on the research involved and the results it yielded, it is suggested that further investigation be undertaken into the possibility of combining the methods and emphasizing learning-based approaches. In this work, a methodology utilising data preprocessing, followed by feature selection and ten-fold cross-validation, is used to obtain a fair assessment. The performance of various classifiers, including Decision Tree, Naïve Bayes, Support Vector Machine (SVM), Random Forest, and Multi-Layer Perceptron (MLP), is assessed using four popular metrics: accuracy, precision, recall, and F1-score. The findings show that the ensemble and neural network outperform traditional classifiers for some data sets. The Multi-Layer Perceptron achieves the best predictive accuracy, while the Random Forest shows consistent and reliable performance. This suggests that machine learning has potential applications that go beyond simply predicting software failure in an early and efficient manner and explores further in the realm of hybrid and learning-focused approaches.

Keywords: *Software Fault Prediction, Machine Language, Neural Network, Software Metrics, Software Quality*

1. Introduction

1.1 Background of the Study

Over the past few years, there has been increasing attention to the use of machine intelligence and data-driven methods for software profiling. Software architectures exhibit complex behaviours, which are hard to identify, predict and categorize anomalies or faults during the development process and run-time. Software profiling data were analysed using machine learning and statistical modelling in Python and R. Both offer a wide range of libraries and features for detecting sophisticated tampering patterns and evaluating models that have been extensively used to analyse and predict faults in software applications [1].

*Corresponding author: Pooja Singh, Department of Computer Science, Al-Falah University, Faridabad, Haryana, India. (research5960@gmail.com)

Fortunately, there are numerous Python libraries for feature selection, data cleaning, and model training, such as scikit-learn, TensorFlow, and Pandas. R is used for its powerful statistical competences, and is useful for testing models, exploring data and making plots to help identify outlines. Even with these tools, it's hard to see how we can easily mark software bugs with them, especially when dealing with a mix of data from various software systems. There is a need for further testing to support their use in practice. Several machine learning methods, including decision trees, SVMs, random forests, KNNs, and neural networks, were employed to identify the largest number of failure points in the software. They looked at memory, CPU usage, various boot paths, logs and so much more for patterns that might be different and could mean trouble. The researchers didn't just pick any method; some might be more effective with one type of data than another.

They created multiple models to classify a part of the code as buggy or not and even how serious the bug might be and how likely it would be. All models were tested on various metrics, including accuracy, precision, recall, and F1 score, to determine the best model. This investigation explores opportunities to enhance the ease with which software problems can be located through data mining and machine learning, enabling more intelligent software construction and testing. Such intelligent software quality control and error-detection techniques are becoming increasingly popular to automate them. Software bugs remain an interesting problem to solve, particularly in sectors where they might be life-threatening, such as healthcare or aviation.

These slips are one of the most common problems for computer scientists and a common annoyance. The fusion of the bio-inspired technology approach, optimization methodologies and machine learning is crucial in the current context of increasingly complex software and the growing presence of real-time systems. In this subject, software design, AI (Artificial Intelligence), information security and systems architecture are introduced and incorporated to enable the building of intelligent and adaptive systems. The application of machine learning to software profiling is expected to yield better defect localisation, more robust anomaly detection, and more proactive debugging methods [4]. This research introduces various enhancements to data mining techniques used to improve automated evaluations, software reliability, and efficiency in the field of fault-tolerant technology.

1.2 Software Fault Prediction

System fault prediction aims to identify code sections that may be affected by faults, using relevant aspects of the software development lifecycle before testing begins. It allows for obtaining the highest quality software with lower expenditure and effort [5]. The main aim of software companies is to provide superior products to their customers. Reliability is an important aspect for end users in evaluating software quality. The aim of software fault prediction is to improve software quality by identifying modules susceptible to rapid failure and classifying them according to specific criteria. They should be detected as early as possible, as the cost of software correction or alteration is far less expensive if detected during the software development process. Predicting software failures is essential in maintaining quality software. At present, there are multiple supervised learning techniques to detect faulty instances. But these training methods just aren't cutting it, as there is a need for smarter ways to spot software bugs early [7]. The aim of software bug prediction is to determine which files or lines of code are likely to cause issues based on the project's essential elements. Typically involves creating a model from project information, such as previous bug records from an older project. The model then attempts to predict where bugs might be found in projects that have not been seen before. The premise of software fault prediction is that if a software product built

in one type of software system experiences faults, then a software product built in a similar system would experience similar faults, provided that it has similar project attributes.

The early, precise handling of software faults is a key step to ensuring software development without bugs and within the defined deadline and budget. It may reduce long-term testing costs [8]. Fault-forecast methodology allows managers to strategically allocate testing resources to modules that may be prone to faults, helping manage costs.

1.3 Machine Learning Algorithms in Software Fault Detection

The detection of software faults is a vital part of software quality control, which is used to find faults or errors in the system that can cause failure. Over the last couple of years, computational linguistic algorithms have shown significant potential for automating and improving this process. While traditional software testing methods are crucial, they can be time-consuming and miss intricate or subtle defects [9]. Machine learning is useful for identifying software bugs, as it can learn from previous software behavior and fault data. Rather than taking chances, it instead documents everything from lines of code, complexity of code, changes while the software is operating, to test coverage of various parts of the system. The machine learning uses all this information to create models which determine which components of the code are likely to be wrong and which are correct.

All of this is fed into machine learning to build systems able to identify possibly buggy and non-buggy code. In this field it's primarily supervised learning, which is a process in which the developers feed the model some examples at the beginning, so it can learn to classify them as defective or not defective. In between, there are common procedures such as Support Vector Machines, Random Forests, Decision Trees, Naive Bayes and Multi-Layer Perceptron's. All of them have been shown to be pretty good defect detectors. The test is to get accurate labelled data to enable the models to learn what is different between correct and incorrect code. What makes it so special is that decision trees provide a clear representation of the features most important for predicting bugs, making them easy to use and understand. SVMs are often used for yes/no (binary) classification, and SVMs are generally used to estimate the risk of the code sections. If you've got very large and intricate datasets, you've got neural networks, exactly for deep learning, that can work with them. They can recognise extremely weak nonlinear patterns that are not recognisable with conventional pattern recognition. Recently, unsupervised learning, such as clustering and anomaly detection, has become more popular, particularly when the labelled dataset is not very large. These techniques detect bizarre or suspicious behaviour or code and make it more apparent that something is wrong. For example, an outlier in software data can be detected using k-means clustering, indicating a fault or unusual software behaviour.

Ensemble methods, such as Random Forests and Gradient Boosting, combine multiple models into a single model. This balances and complements the weaknesses of one model with the strengths of the other. If software changes occur, machine learning models can also be updated to incorporate the latest information, keeping them up to date with the latest software and bug fixes. With machine learning defect analysis, teams have a much better idea of where to focus their attention on a codebase. It therefore helps developers focus their testing and debugging efforts on the relevant points, saving them a lot of time and effort and reducing the physical burden, as they can now rely on automated defect prediction. Overall, machine learning has emerged as a key tool for detecting software problems. They can process information and make accurate forecasts, which help proactively identify and resolve software issues. As software becomes more complex, machine learning will be even more vital to keep software stable, safe, and performing.

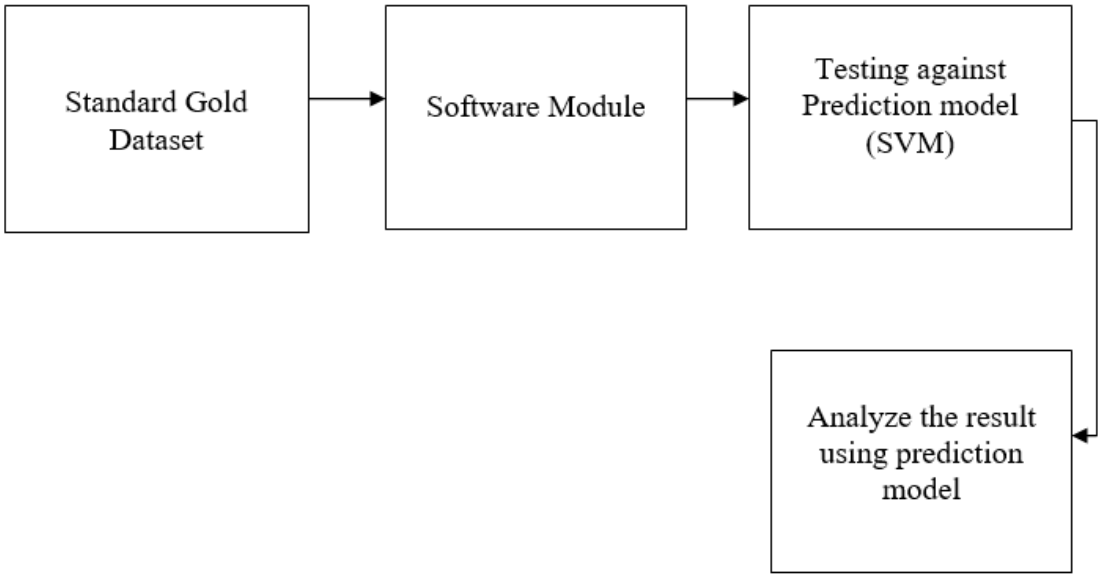

Figure 1: Application-based Classification using ML Algorithm for Gold Dataset

1.4 Neural Network Algorithms in Software Fault Detection

A neural network is an advanced type of data modelling tool that can learn and describe complex input-output relationships. It is possible for neural network technology to execute “intelligent" operations similar to those carried out by the human brain. It learns the information and then stores it in the synaptic weights between different interneurons. A wide range of problems can be solved using neural networks, such as breast cancer detection and classification of satellite images [12].

1.4.1 Characteristics of Neural Networks

- Neural networks have a mapping function that can link input patterns to their corresponding output patterns. Neural networks learn by watching, imitating other people's behaviors.
- The training of a neural network architecture can be performed on known instances of a problem before testing the architecture on the inference of unknown instances of the problem. As a result, people can recognize things that they weren't aware of before.
- Neural networks can make generalizations about their conclusions. They can predict what may happen next based on what has happened before.
- Neural networks are very reliable and fault tolerant. Thereby, they might be able to get complete patterns from incomplete, partial, or distorted patterns.

One of the characteristics of neural networks is that they can process information concurrently, quickly, and in a decentralized way.

1.4.2 Abstract Model of Neural Network

- Feed – Forward Artificial Neural Network

Feed-forward ANNs only have one direction of information flow between the input and the output (figure 2). No interlayer feedback loops - output of one layer does not affect any other layer. One of the basic feed-forward ANN architectures is a chain of simple networks between inputs and outputs (I/O). Many different kinds of pattern recognition problems may make use

of these techniques. Such organizational schemas can be considered as from the bottom or top, depending on the source [13].

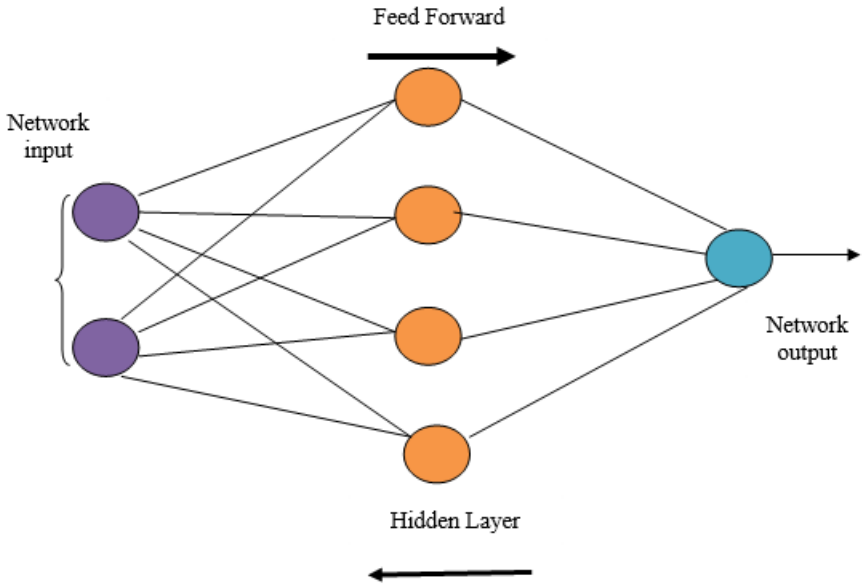

Figure 2: Neural Network

- Feed – Backward Artificial Neural Network

Feedback networks can carry both forward and backward transmission owing to their loops. The aim of feedback networks is to keep things in constant flux until they reach equilibrium. It means they must maintain input stability, which means they must be in balance until a new balance is reached. Feedback designs, also referred to as recurrences, are more commonly used for feedback linkages in one-layer organisations. Another name is interaction.

1.4.3 Layers in Neural Network

Generally, an artificial neural network consists of an input layer, a hidden layer, and an output layer, with the hidden layer connected to the output layer.

- The activity of the input units denotes the raw data that is input to the network.
- The output of each hidden unit depends upon the output of the input units and the weights on the connections between the input and hidden units.
- The hidden units should be active and the weights connecting the hidden and output units should be the same size in order for the output units to work properly.

This simple network architecture plays a crucial role, enabling the hidden units to form their own representations of the input. The representations of hidden units can be trained by adjusting the weights of the input-layer units. There are single-layer and multi-layer. Most of the organisational structure is unilayer, and is all interrelated. This organization has more computing power than multi-tiered hierarchical organizations. In multi-layered networks, different naming standards are used from those used in the majority of networks.

1.4.4 Multilayer perceptron

A feed-forward neural network model is used to process input data and produce results. The block diagram of a three-layer network is shown in Figure 2 below. It contains an input, hidden, and output layer. Nodes that are not connected are called input neurons in a network. The term

"output neuron" describes a node that does not have any connections to other nodes. The term "hidden neurons" refers to nodes that are neither input nor output nodes [14]. In the hidden layer, perceptrons have sigmoidal functions, whereas in the input layer, they have linear functions. Below are the calculations carried out at each tier.

- **Input Layer Computation**

Without doing any weighted summation or thresholding, the input layer just gathers data and passes them on to the next layer. This indicates that the information which is input into the network is manifesting in the neurons of the input layer. If the activation function used is linear (f = tan(=1)), the input to the input layer equals the output from the input layer. By making use of only one dataset

$$\{O\}_I = \{I\}_I \quad (1)$$

There are synapses between the ith input neuron and the jth hidden neuron, with the weight between them being Wij. The weighted sum of the output from the input neuron is used to obtain the input to the kth hidden neuron (I_{Hk}).

$$I_{Hk} = W_{1k}O_{I1}+W_{2k}O_{I2}+\ldots..+ W_{Ik}O_{I1} \quad (2)$$

Weights are represented by the matrix [W], and the output of the input layer is given as the input to the hidden neuron below with k=1,2,3...m.

$$\{I\}_H=[W]^T\{O\}_I \quad (3)$$

- **Hidden layer computation**

The weights that connect the input and hidden units, as well as the activity of the neurons in the input layer, determine the activity of the neurons in the hidden layer. The output of the kth hidden neuron could be represented by the sigmoid function, which is:

$$O_{Hk} = \frac{1}{1+e^{-\gamma(I_{Hk}-\Theta_{Hk})}} \quad (4)$$

If O_{Hk} is the output from the kth hidden neuron, I_{Hk} is the input to the kth hidden neuron, and H_k is the threshold of the pth neuron is used, then the result is:

$$I_{Oq}=W_{1q}O_{H1}+ W_{2q}O_{H2}+\ldots+ W_{mq}O_{Hm} \quad (5)$$

The information from the output of the k^{th} hidden neuron is sent to the input of the output neuron, and the output neuron processes the information. The following is the evaluation of the input to the qth output neuron:

Also, in this instance, the subscripts O and H denote the numbers of output and hidden layer neurons, respectively, and the numbers 1, 2, 3, ..., n denote the number of each type of neuron. The input to the output neuron is denoted as, and the weighted matrix between the hidden neurons and output neurons is:

$$[W].\{I\}_O=[W]^T\{O\}_H \quad (6)$$

- O**utput layer computation**

The activity of neurons in the output layer depends on the activity of the hidden layer and the difference between the weights of the hidden layer and the output layer. The output of a q^{th} output neuron is given by sigmoid function.

$$O_{oq} = \frac{1}{1+e^{-\gamma(I_{0q}-\Theta_{0q})}} \quad (7)$$

The output Oq at the output neuron q^{th} in is given by equation 7.

The backpropagation method is used to train a multi-layer feedforward neural network, consisting of many layers. The algorithm always generates the set of weights to predict class labels. Network predictions are compared to actual target values using backpropagation technique [15].

Ideally, one would like to have the class label information in the training set. By adjusting the weights of each learning pair, the mean squared error, a measure of how close a network's prediction is to the actual outcome, is minimized. The adjustments begin in the output layer and propagate back through the other hidden layers down to the top-most hidden layer. Equalization of the weights and completion of the learning process may be thought to be the conditions of convergence.

1.2 Problem Statement

Despite comprehensive study on software fault prediction, attaining continuously accurate predictions across varied datasets continues to be a problem. Numerous current models have challenges like overfitting, inadequate generalization, susceptibility to unbalanced data, and restricted interpretability. Moreover, not one specialized machine learning technique achieves optimum performance across all categories of software projects. To recognize the efficiency of the various fault prediction procedures, you need to assemble a set of classifiers, apply them to the similar sets of data, and use a "fair" technique of evaluation. You can individually determine which ones are solid and can withstand new data by seeing which singles they are.

The research purposes to achieve the following objectives:

- To evaluate the effectiveness of machine learning and neural network processes on software defect prediction.
- To evaluate and comparison the effectiveness of different categorization methods that rely on reference databases.
- To determine the most suitable models for prediction of fault prone system mechanisms.

1.4 Scope and Significance

This study will explore supervised machine learning and AI techniques applied to real software metrics datasets. These datasets are from real projects and not toy examples. The whole idea is to get a more in-depth look at the relative performance of various classifiers, both for research and industry. Let's tell you why a data-driven fault prediction isn't a buzzword, but it is a valuable tool for you. Armed with these insights, software developers, testers, and project managers will be better equipped to make informed decisions and proactively address bugs before they become issues.

1.5 Organization of the Paper

Here’s what’s next: Section 2 takes a deeper dive into the existing research on software failure prediction. Section 3 covers how I approached the study, what datasets I picked, and how I set up the experiments. Section 4 goes through what happened in those experiments and compares the different approaches. Section 5 wraps things up and points to what still needs exploring.

2. Literature Review

Spotting software problems early on by profiling cuts down on risks tied to software crashes. Profiling lets you perform dynamic analysis, which complements traditional static code checks. That means you’re far more likely to catch tricky bugs that are hidden deep in the code. If you look at the research on software profiling, the big themes are fault detection, defect prediction,

quality assurance, and machine learning. All of these areas play a huge role in ensuring software is not only accurate but also maintainable.

2.1 Role of Machine Learning in Software Fault Prediction

We've known for a while that predicting program failures makes software more reliable and saves maintenance costs. Early research mainly focused on statistical and regression models based on factors such as software size and complexity. But these models struggled with messy, real-life data, especially when things got nonlinear or a little too complex. So, they couldn't always predict as well as people hoped.

Tasneem, N. et al. (2025) delineated recent breakthroughs in extracting scholarly coding data, outlined obstacles pertinent to this domain, highlighted successful implementations, and proposed new directions for additional study. Employing proven data mining methods, academics and practitioners have been investigating the importance of this valuable data to enhance project maintenance and build higher-quality software applications on time and on budget. They presented techniques for predicting practical issues. Client satisfaction drops, manufacturing and maintenance expenses go up, and software issues arise from defective components.

Utilizing information mining methods and a Multilayer Perceptron Neural Network (MPNN), Charan, S. et al. (2024) proposed a software fault system. To model applications for fault-proneness forecasting, a comparison was conducted, and a sophisticated artificial intelligence architecture based on multi-layered convolutional neural networks (MPNNs) was examined. This collection contains software measurements retrieved from the NASA Measurement Data Program (MDP).

Zhou, X. et al. (2024) introduced an innovative method, Multi-Objective Feature Selection (MOFES). PROMISE, an innovative way for managing tasks, sets itself apart from comparable strategic methods throughout the thorough assessment process. The results clearly show that this new approach can identify key features, boosting performance and raising expectations. It strikes a sweet spot between getting good results and keeping everything cost-efficient. PROMISE proves it works well and holds up under pressure, making it a solid option for improving production and success across all kinds of jobs and projects.

2.2 Research Gaps

While there's certainly been a lot of suggestions on how to categorize software failures, no one has gotten it right. Many studies are limited to a single data set or to only a few metrics. This is an active field of research, and this project addresses current gaps by conducting a comprehensive comparison of several neural networks and classification methods across multiple datasets.

3. Methodology

3.1 Dataset Description

You can look at datasets in all sorts of ways: size, where they come from, time stamps, color you name it. These features, or variables (sometimes called fields or attributes), usually show up as columns in a dataset. In the field of machine learning, a feature is simply any measurement of what you're studying. In this study, four data sets were utilized which are available to everyone. These are from the PROMISE academic software defect prediction database, which is pretty popular in the software defect prediction world. The data sets (PC1, PC2, KC1, and KC3) used are from NASA's repository [21-23].

To see how well their methods work, the researchers tested them on real software datasets. The datasets from NASA, typically used in fault prediction studies, were obtained from the PROMISE archive. Table 1 provides details and the number of cases for each dataset.

Table 1: Dataset Specifications Featuring Defective and Non-Defective Instances

Dataset	Attributes	Instances	Faulty Instances	Non-Faulty Instances
Dataset (PC1)	22	1109	77	1032
Dataset (PC2)	37	745	16	729
Dataset (KC1)	22	2109	326	1783
Dataset (KC3)	40	194	36	158

This research used four datasets PC1, PC2, KC1, and KC3 obtained from the NASA Metrics Data Program and PROMISE repository, to assess software failure prediction methodologies. These databases include software metrics drawn from real NASA software projects, and researchers often use them to predict faults with machine learning.

3.2 Research Design

The first step in this study was to obtain the data to fill in missing values, normalise the numbers, and select the features to include. Data values were imputed statistically to ensure reliability of the set. This type of preprocessing improves the model's accuracy and stability.

The research takes an experimental approach, testing machine learning and neural network models for predicting software faults. We now proceed step by step: (1) Prepare the data, (2) train the models, and (3) test the performance of the models. The study compares five different classifiers: Support Vector Machine (SVM), Random Forest (RF), Decision Tree (DT), Naive Bayes and Multi-Layer Perceptron (MLP). A detailed description of the methodology is provided in the following section, and the methodology is illustrated in Figure 3. There are three key steps in the process:

Phase I - Data Preparation:

In this part, the main emphasis is on cleaning the data and ensuring it is ready for classification – in other words, cleansing, normalizing and organizing features. We then used PCA to reduce the number of features while preserving the most relevant information.

Phase II - Model Evaluation Using Python:

Here we have implemented and tested the selected classifiers using Python. Cross-validation is important; it lets us see if the prediction model really works. In particular, tenfold cross-validation, which means we divide the data into 10 parts. This approach provides more accurate performance measures than a single split. A few additional numeric features were added to improve the classification process.

Phase III - Performance Metrics Assessment:

Finally, we closely examined how each model performed, employing the re-substitution method (also referred to as the self-constitution test), in which the training data is used for testing. In this case, we emphasized the classification accuracy and examined a few metrics

including specificity, sensitivity, precision and the F1 score, all based upon the Gold Dataset standardized during the study [24-25].

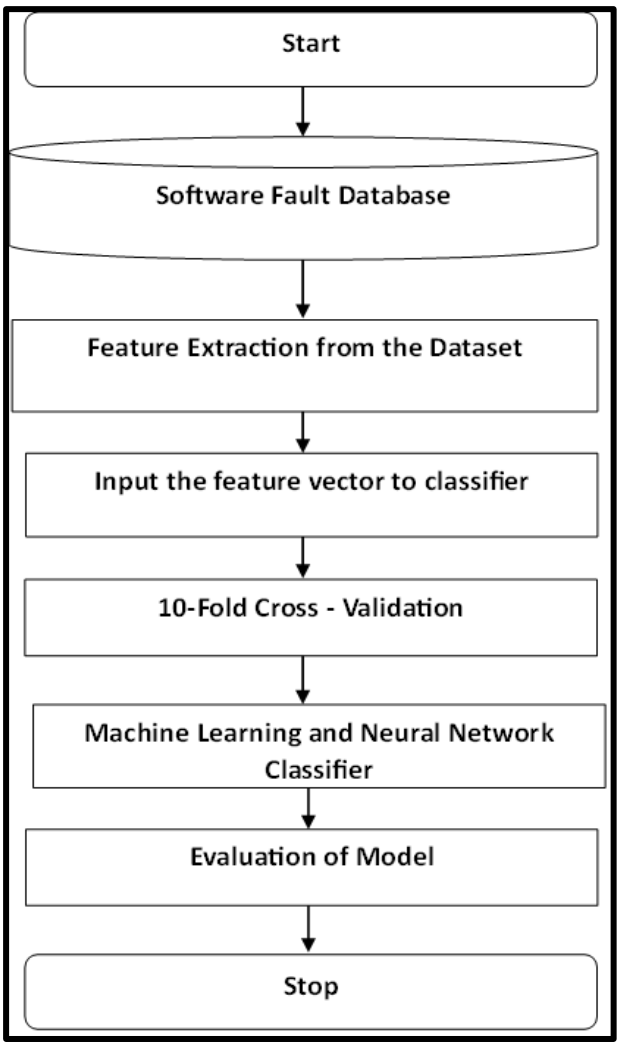

Figure 3: Proposed workflow diagram for Software fault Detection

4. Results and Discussion

In the research of machine learning for software issue diagnosis, there is a clear trend that the more complex a software model is, the better the features it possesses are defined, the higher the model's performance tends to be. The aim was straightforward: To move software components into one of two categories, “fault-prone” or “unaffected”, based on metrics extracted from the software code. Decision Trees performed very well, achieving 97% accuracy. They don't perform fancy math; they only apply simple decision rules. When the input is well-structured, this is sometimes sufficient. The Naive Bayes model achieved 97% accuracy, which is quite impressive for such a simple model [26, 27]. It did so by leveraging the most relevant software features and pre-processing the data, demonstrating that it is not necessary to have a complicated model when you choose the right data. Random Forest went one more step. It was not only correct (also 97% correct), but much more consistent, as it pulled in predictions from a knaps of decision trees. This ensemble method was shown to be more effective at avoiding overfitting and to perform well across various cases, making it a strong choice for real-world software defect prediction. Much like a neural network, MLP can identify non-linear relationships in software measurements, but it requires good training data. Here is a high score, demonstrating the power of deep learning, when you have enough good examples.

SVM? Not so great here. It only achieved 39% accuracy. SVMs tend to struggle if the data isn't scaled well, or if the separation between classes isn't nice and neat. With some adjustments, such as a special kernel or improved feature engineering, it simply cannot catch up. Table 2 summarises the accuracy, precision, recall, and F1-score for each model. From these results, it is evident that ensemble methods and deep learning models are the most suitable models for software defect prediction; they achieve high accuracy and reliability results [29-30].

Table 2: Comparison-Based Evaluation of Machine Learning Framework Classification

Proposed Model	Dataset	Accuracy	Precision	Recall	F1-Score
Decision Tree (DT)	PC1	0.97	0.98	0.93	0.94
	PC2	0.97	0.95	0.96	0.97
	KC1	0.97	1.0	1.0	1.0
	KC3	0.97	0.94	0.94	0.94
Random Forest (RF)	PC1	0.97	0.94	0.96	0.97
	PC2	0.97	0.97	0.95	0.96
	KC1	0.97	1.0	1.0	1.0
	KC3	0.97	0.97	0.96	0.97
Naïve Bayes (NB)	PC1	0.97	0.97	0.96	0.96
	PC2	0.97	0.95	0.96	0.97
	KC1	0.97	0.99	0.98	0.98
	KC3	0.97	0.95	0.98	0.97
Support Vector Machine (SVM)	PC1	0.39	1.0	0.21	0.31
	PC2	0.39	0.13	0.18	0.1
	KC1	0.39	0.95	0.56	0.71
	KC3	0.39	0.49	1.0	0.43
Multi-Layer Perceptron (MLP)	PC1	0.99	0.98	0.98	0.98
	PC2	0.99	0.99	0.99	0.99
	KC1	0.99	1.0	1.0	1.0
	KC3	0.99	0.98	0.99	0.98

The Table 2 data clearly shows that deeper convolutional models, such as Support Vector Machine (SVM), Random Forest (RF), Decision Tree (DT), Naive Bayes, and Multi-Layer Perceptron (MLP), provide significant advantages in software fault detection. To detect faults very accurately, designs need to be very good at capturing complex, multi-layered knowledge. Certainly, simple models lay the foundation, but to achieve high levels of accuracy requires something more advanced. Of all these, the Multi-Layer Perceptron (MLP) is the most remarkable: it is robust and reliable yet able to detect software defects. Below is a chart that (visually) presents it all.

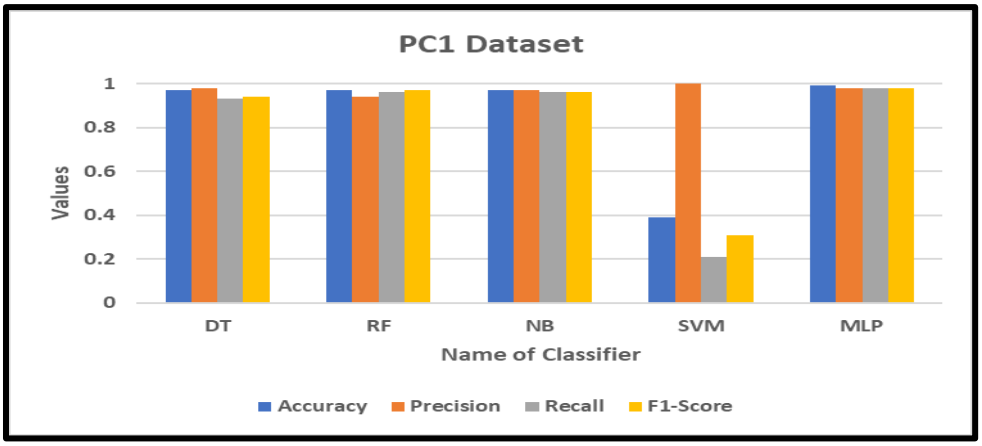

Figure 4: Comparing the Outcome of Classifiers for the Dataset PC1

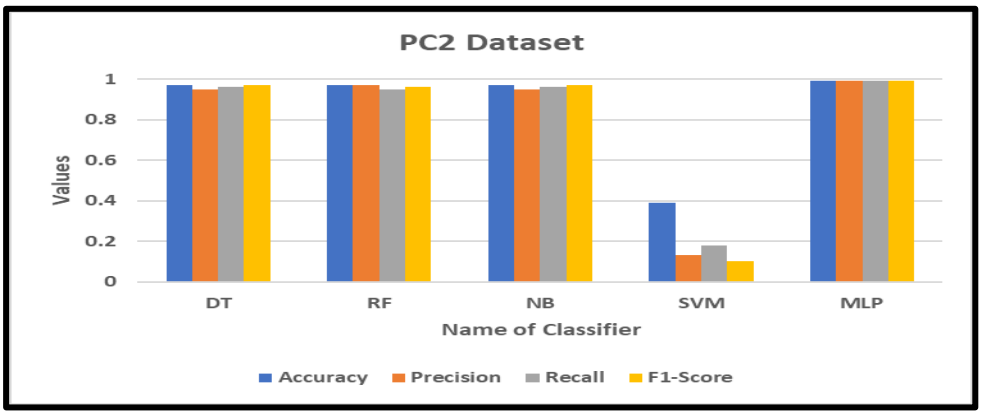

Figure 5: Comparing the Outcomes of Classifiers for the Dataset PC2

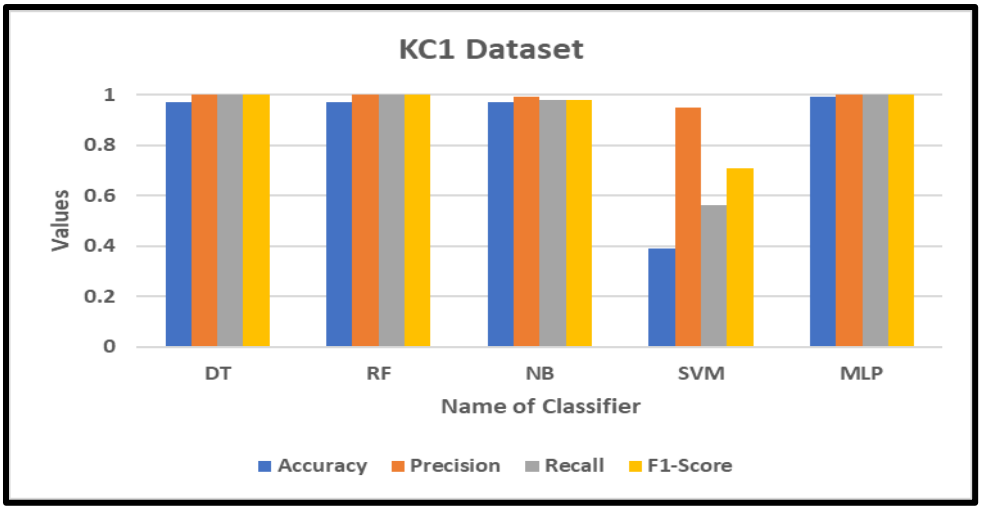

Figure 6: Comparing the Outcomes of Classifiers for the Dataset KC1

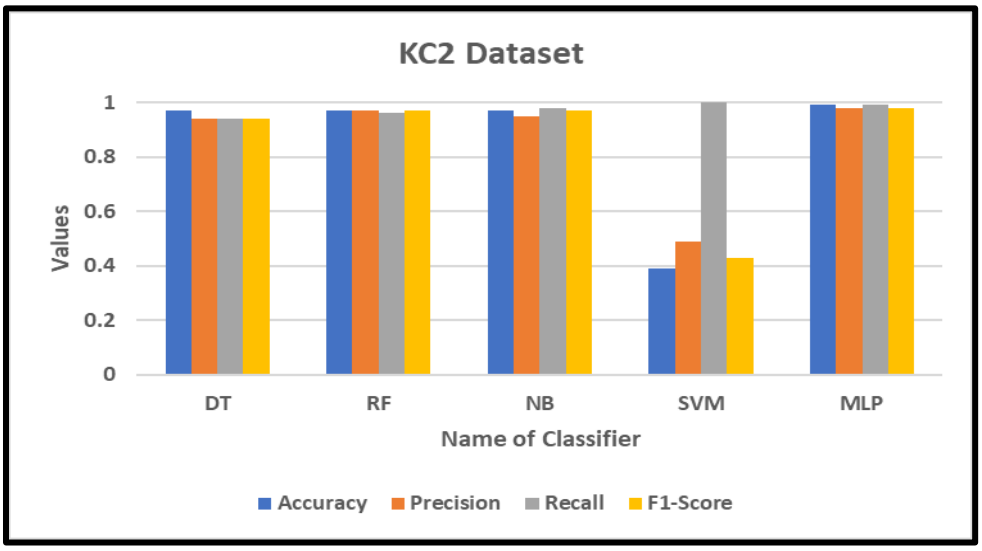

Figure 7: Comparing the Outcomes of Classifiers for the Dataset KC2

This section illustrates how we used a complete process to address software defect detection: first, standardizing the data, then constructing the classifiers, and finally testing the classifiers to see how well they would perform on the data. We examined the performance of five models, i.e. Support Vector Machine (SVM), Random Forest (RF), Decision Tree (DT), Naive Bayes, and MLP, in detecting software flaws.

5. Conclusion and Future Growth

5.1 Conclusion

It goes into the details of how machine learning and neural networks can be used to predict software failures, using NASA benchmark datasets. Surprisingly, ensemble and neural network models are superior to the traditional classifiers in terms of accuracy and flexibility. The findings suggest that this rigorous fault-prediction methodology, based on data analysis, can take software to the next level.

The datasets are readily available, but not quite up to date with current software development practices or software development dynamics. Research could go deeper into deep learning, hybrid approaches and transfer learning to stay up to date with today's software for next steps. Integrating dynamic software measurements, real-time data, and explainable AI methodologies

might augment the efficacy and comprehensibility of defect prediction mechanisms of industrial contexts.

References

[1] Shelke, C., Mandale, A., Anjimoon, S., Asha, V., Nijhawan, G., & Dhanraj, J. A. (2025). Optimized Machine Learning Techniques for Software Fault Prediction. *Natural Language Processing for Software Engineering*, 207-219.

[2] Alsangari, B., & Bircik, G. (2023, June). Performance Evaluation of various ML techniques for Software Fault Prediction using NASA dataset. In 2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA) (pp. 1-7). IEEE.

[3] Pavana, M. S., Pushpalatha, M. N., & Parkavi, A. (2021, October). Software fault prediction using machine learning algorithms. In International Conference on Advances in Electrical and Computer Technologies (pp. 185-197). Singapore: Springer Nature Singapore.

[4] Liu, J., Pan, C., Lei, F., Hu, D., & Zuo, H. (2021). Fault prediction of bearings based on LSTM and statistical process analysis. Reliability Engineering & System Safety, 214, 107646.

[5] Siddiqui, T., & Mustaqeem, M. (2023). Performance evaluation of software defect prediction with NASA dataset using machine learning techniques. International Journal of Information Technology, 15(8), 4131-4139.

[6] Jakhar, A. K., & Rajnish, K. (2018). Software Fault Prediction with Data Mining Techniques by Using Feature Selection Based Models. International Journal on Electrical Engineering & Informatics, 10(3).

[7] P. Tyagi, A. K. Bindal and D. Srivastava, "A Fusion-Driven Deep Learning Framework for Robust and Scalable Face Recognition in Biometric Systems," *2025 International Conference on Data, Energy and Communication Networks (DECoN)*, Bhopal, India, 2025, pp. 1-6, doi: 10.1109/DECoN67170.2025.11447814.

[8] Basha, N., Nounou, M., & Nounou, H. (2018). Multivariate fault detection and classification using interval principal component analysis. Journal of computational science, 27, 1-9.

[9] Rathore, S. S., & Kumar, S. (2019). A study on software fault prediction techniques. Artificial Intelligence Review, 51, 255-327.

[10] Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. International journal of advanced computer science and applications, 9(2).

[11] Tantithamthavorn, C., Hassan, A. E., & Matsumoto, K. (2018). The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Transactions on Software Engineering, 46(11), 1200-1219.

[12] Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., ... & Zimmermann, T. (2019, May). Software engineering for machine learning: A case study. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 291-300). IEEE.

[13] Batool, I., & Khan, T. A. (2023). Software fault prediction using deep learning techniques. Software Quality Journal, 31(4), 1241-1280.

[14] Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., ... & Soomro, T. R. (2021). Software defect prediction using ensemble learning: A systematic literature review. IEEe Access, 9, 98754-98771.

[15] Arasteh, B., & Branch, T. (2018). Software fault-prediction using combination of neural network and Naive Bayes algorithm. *J. Netw. Technol*, *9*(3), 95.

[16] Albreiki, B., Zaki, N., & Alashwal, H. (2021). A systematic literature review of student'performance prediction using machine learning techniques. Education Sciences, 11(9), 552.

[17] Tyagi, P., Bindal, D.A., & Srivastava, D. (2025). An Optimized Feature-Level Fusion Framework for Multimodal Biometric Authentication Using ML Classifiers. *International Journal of Environmental Sciences*.

[18] Tasneem, N., Zulzalil, H. B., & Hassan, S. A. (2025). Enhancing Agile Software Development: A Systematic Literature Review of Requirement Prioritization and Reprioritization Techniques. *IEEE Access*.

[19] Charan, S., Sinha, S., Sujan, S. M., & Nayak, U. A. (2024, May). Accuracy Prediction of Ensemble Deep Learning Model through Software Defect Prediction. In *2024 Second International Conference on Data Science and Information System (ICDSIS)* (pp. 1-8). IEEE.

[20] Zhou, X., Yuan, W., Gao, Q., & Yang, C. (2024). An efficient ensemble learning method based on multi-objective feature selection. *Information Sciences*, *679*, 121084.

[21] Karim, S., Warnars, H. L. H. S., Gaol, F. L., Abdurachman, E., & Soewito, B. (2017, November). Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset. In *2017 IEEE international conference on cybernetics and computational intelligence (CyberneticsCom)* (pp. 19-23). IEEE.

[22] Zhang, B., Li, Y., & Chai, Z. (2022). A novel random multi-subspace based ReliefF for feature selection. *Knowledge-Based Systems*, *252*, 109400.

[23] Ayon, S. I. (2019, May). Neural network-based software defect prediction using genetic algorithm and particle swarm optimization. In *2019 1st International conference on advances in science, engineering and robotics technology (ICASERT)* (pp. 1-4). IEEE

[24] Srivastava, D., Mall, S., Singh, S. P., Bhatt, A., Kumar, S., & Soni, D. (2024). Deep ensemble model for sequence-based prediction of PPI: Self improved optimization assisted intelligent model. Multimedia Tools and Applications, 83(26), 68135-68154.

[25] Sharma, V., Kumar, L., & Srivastava, D. (2023). Machine Learning-Based Prediction of Users' Involvement on Social Media. In Advanced Applications of NLP and Deep Learning in Social Media Data (pp. 151-170). IGI Global Scientific Publishing.

[26] Goyal, D., Choudhary, A., Pabla, B. S., & Dhami, S. S. (2020). Support vector machines based non-contact fault diagnosis system for bearings. Journal of Intelligent Manufacturing, 31(5), 1275-1289.

[27] Iqbal, A., Aftab, S., Ali, U., Nawaz, Z., Sana, L., Ahmad, M., & Husen, A. (2019). Performance analysis of machine learning techniques on software defect prediction using NASA datasets. International Journal of Advanced Computer Science and Applications, 10(5).

[28] Kaur, G., Pruthi, J., & Gandhi, P. (2023). Decision tree regression analysis of proposed metric suite for software fault prediction. *SN Computer Science*, *5*(1), 69.

[29] Mustaqeem, M., Mustajab, S., & Alam, M. (2024). A hybrid approach for optimizing software defect prediction using a grey wolf optimization and multilayer perceptron. *International Journal of Intelligent Computing and Cybernetics*, *17*(2), 436-464.

[30] Juneja, S., Nauman, A., Uppal, M., Gupta, D., Alroobaea, R., Muminov, B., & Tao, Y. (2024). Machine learning-based defect prediction model using multilayer perceptron algorithm for escalating the reliability of the software. *The Journal of Supercomputing*, *80*(7), 10122-10147.