Software Quality Analysis Based on Selective Parameters Using Enhanced Ensemble Model

Rakhi Singh
•
Department of Computer Science & Engineering, Shobhit Institute of Engineering & Technology (Deemed-to-be University), Meerut, India
research.rs2k11@gmail.com

Mamta Bansal
•
Department of Computer Science & Engineering, Shobhit Institute of Engineering & Technology (Deemed-to-be University), Meerut, India
mamta.bansal@shobhituniversity.ac.in

Surabhi Pandey
•
Department of ICT & e-Governance, Indian Institute of Public Administration, New Delhi, India
dr.pandeysurabhi@gmail.com

Abstract

Software Quality Analysis refers to the process of evaluating and assessing the quality of software products or applications. It involves analyzing various aspects of the software to determine its level of quality, identify potential issues or defects, and make informed decisions to expand the software's overall quality. There are investigated different software quality models based on machine learning algorithms. Nevertheless, the explored approaches have an inconsistent understanding of the software product quality and high complexity. This research presents an enhanced ensemble model (EEM) that involves Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Extreme Learning Machine (ELM) to assess the optimal outcomes. This model performance is computed based on multiple selective parameters namely functional suitability and maintainability of the software and compared along with several algorithms namely Decision Tree, Random Forest, AdaBoost, and Naive Bayes. The outcome of this ensemble model demonstrates that it offers highly accurate results for software validation, verification, and overall product development process to analyze the functional suitability as well as software maintainability. The measured accuracy on Decision Tree, Random Forest, Naive Bayes, AdaBoost, and EEM is found 92.08%, 93.35%, 94.50%, 95.60%, and 99.14%, respectively

Keywords: Software Quality Analysis, Enhanced Ensemble Model (EEM), Decision Tree, Random Forest, Naive Bayes, AdaBoost.

1. Introduction

The process of evaluating the quality and efficiency of software systems is known as Software Quality Analysis (SQA), commonly referred to as Software Quality Assurance or Software Testing [1]. To find flaws, errors, or variations from expected behavior in software products, systematic activities, and procedures are used. Software quality analysis' main goal is to make sure that the

entire software product works properly, complies with specifications, and offers a desired user experience. Activities like test planning, tracking of defects, test designing, execution of tests, reporting of tests, and other testing kinds including performance testing, security testing, and usability testing are all a part of this process. Organizations may reduce the chance of problems in production and produce high-quality software to satisfy user expectations by undertaking software quality analysis [2]. This allows organizations to discover and fix errors early in the development lifecycle. There are available different software quality analysis techniques such as static analysis and many others. To estimate and promise the quality of software systems, a variety of approaches and techniques are used in software quality analysis. Following are some examples of frequently used software quality analysis techniques [3]:

1.1. *Static Analysis:*
In this technique, the program is not run; instead, the source code, design documentation, and other artifacts are examined. Code concerns such as coding mistakes, coding standards violations, security flaws, and performance inefficiencies are all examined using static analysis tools.

1.2. *Dynamic Analysis:*
Dynamic analysis aids in the discovery of runtime problems such as memory leaks, performance bottlenecks, and functional flaws.

1.3. *Regression Testing:*
Regression testing is done to make sure that changes to the program do not result in the introduction of new flaws or affect current functioning.

1.4. *Performance Testing:*
It helps in the detection of performance bottlenecks, resource use problems, and response time lags. Techniques for performance testing include load, stress, and endurance testing [4].

1.5. *Security Testing:*
This method is utilized to find weaknesses and possible attack entry points. It also utilizes the strategies including penetration testing, vulnerability scanning, and security code reviews.

1.6. *Usability Testing:*
Usability testing looks at the user interface (UI) and user experience (UX) of the program to make sure that it is simple, effective, as well as satisfies user expectations.

1.7. *Code Reviews:*
Code reviews entail systematically going over source code to look for coding flaws, performance problems, maintainability difficulties, and adherence to coding standards.

1.8. *Test Automation:*
Test automation is the process of automating time-consuming and repetitive testing operations using specialized tools and frameworks.

There have been investigated multiple ways which introduce how can improve software quality analysis and according to that the study of software quality may be greatly enhanced by combining useful methods and strategies. The use of thorough testing methods at every stage of the software development lifecycle is a critical task [5]. By integrating unit tests, integration tests, regression tests, system tests, as well as acceptance testing, software developers may ensure a

thorough assessment of the quality of their product. Each testing level has a distinct function and is capable of spotting many kinds of flaws, enabling a more thorough assessment. Additionally, the use of test automation is crucial for improving software quality analysis. The formulation and execution of automated tests are made possible by automated testing tools and frameworks, which enhance test coverage, lowers human error rates, and boost the general effectiveness of testing procedures [6]. Software problems may be swiftly found and fixed by developers with the capacity to run automated tests frequently, speeding up the resolution process. The examination of software quality is also improved by code reviews and inspections. Regular code reviews enable engineers to see possible problems before they become serious. Developers can find coding errors, confirm that coding standards are being followed, and pinpoint opportunities for development by carefully studying the source code. By taking proactive measures, the software's overall quality is improved and risks are reduced [7]–[9].

J. Mona et al. in [10] explored the increasing need for software applications due to recent technological advancements. The demand for software products has significantly grown across various industries. To ensure the increase of the software industry, it is decisive to create high-quality software and maintain its reputation among users. However, implementing quality assurance standards to instill trust in consumers can be challenging. The software development team often views quality assurance as a time-consuming and documentation-intensive process that adds little value to the client. It is also stated that determining the quality of software can be difficult as it depends on individual perspectives and interests. The study also mentions the use of soft computing-rooted machine learning techniques to address quality as well as assurance in software engineering, intending to build an effective framework to predict software faults. The drawback of this method is that it does not provide specific examples of recent technological breakthroughs or discuss any particular challenges faced in quality assurance or software development. It presents a general overview of the importance of software quality but lacks specific details and concrete examples.

E. Rashid et al. [11] discussed the use of case-based reasoning (CBR), a kind of machine learning, as an expert system to forecast software quality. The major goal of this study is to reduce software expenses. The paper introduces a novel idea of building a knowledge base (KBS) for CBR, which includes new problems and their solutions. The research also aims to reduce maintenance costs by removing duplicate records from the KBS. Four similarity functions are used for error prediction. Case-based models, such as the developed case-based reasoning model, are considered effective when defining rules for a problem domain is challenging. On the other side, one of the drawbacks of using machine learning for software quality analysis is the requirement for large amounts of labeled data. Machine learning algorithms typically perform better when trained on a large and diverse dataset. However, obtaining labeled data for software quality prediction can be challenging and time-consuming.

Research Questions:

- How to improve the software quality analysis with selective parameters and an ensemble model?
- To evaluate the performance metrics using diverse datasets.

2. LITERATURE REVIEW

T. Sharma et al. in [12] discussed the importance of predicting software faults before the testing phase to improve source provision and ensure high-quality software development. Different algorithms are used to address this issue, and various analytical frameworks have been built to classify software modules as defective or non-defective. However, despite the advancements in

software development, poor prediction performance is still observed in many studies. This is attributed to challenges such as redundancy, correlation, irrelevant features, missing data, and an unbalanced distribution between classes with faults and classes without faults in the software data. To tackle these challenges, researchers globally have turned to ensemble different techniques, which have shown certain enhancements in fault estimation performance. This research critically analyzes ensemble-rooted machine-learning methods explored for software fault forecasting from 2018 to 2021. The paper aims to gain a deeper understanding of why hybrid frameworks still exhibit poor performance on accessible datasets. The drawback investigation is that it relies on meta-learning and classifiers for software defect prediction. While meta-learning can be effective in adapting to different datasets and improving generalization, it may face limitations when applied to software defect prediction.

K. Bashir et al. [13] explored that software quality analysis aims to develop forecasting models utilizing the fault data as well as software metrics to identify potentially faulty program modules. This helps in optimizing resource allocation as well as utilization. Nevertheless, the strength of the classifiers and the quality of the data both have an impact on how accurate these prediction models are. The effectiveness of classification models might be jeopardized by problems with data quality such as an excessive number of dimensional imbalances in classes, and the inclusion of diverse noise in software fault data. To improve software development procedure (SDP) models. This study presents a novel and unified framework. The system uses Data Sampling (DS), Iterative-Partition Filter (IPF), and Ranker Feature Selection (FS) approaches to handle the issues of high dimensionality, and imbalances in classes, respectively. The outcome validates the effectiveness of this suggested framework for SDP, indicating that it helps improve the accuracy as well as performance of fault prediction models. The drawback of the proposed technique is that supervised learning requires labeled training data, which may be expensive and time-consuming. Collecting and labeling a comprehensive dataset of software defects can be challenging, especially for large-scale software systems. Limited or incomplete training data can lead to biased or inaccurate predictions, reducing the reliability of the defect prediction framework.

M. Rawat et al. [14] stated that despite thorough planning, extensive documentation, and proper process control throughout software development, it is unavoidable to encounter certain defects. These defects can negatively impact the quality of the software and potentially lead to failure. In today's highly competitive environment, it is crucial to actively work towards controlling and minimizing software engineering flaws. However, these initiatives need time, money, and resources. This paper aims to identify the root causes of these defects and provide suggestions for improving software quality and productivity. Additionally, the paper demonstrates the implementation of various defect prediction models that have resulted in a decrease in the number and severity of defects. The main drawback of this is the potential for inaccurate predictions. While these models aim to identify and predict software defects, there is still a degree of uncertainty involved. The models rely on historical data and patterns to make predictions, which means they may not account for unique or unforeseen circumstances that could lead to defects. Additionally, the accuracy of these models heavily relies upon the quality as well as the relevance of the utilized datasets for training. If the dataset is incomplete, biased, or outdated, then software quality prediction may not be reliable.

S. Yamada [15] investigated that the assessment of software reliability holds significant importance in efficiently developing high-quality software products. This paper focuses on quantitative measurement as well as the evaluation of software product reliability. The methods discussed in this work are centered on nonhomogeneous Poisson processes used in the software product consistency advancement models built in Japan. These models aim to provide a more realistic depiction of the software error-detection and failure-occurrence processes in the test phase of software product expansion by refining the underlying assumptions. The paper provides an

outline of the existing software reliability growth models (SRGMs) as well as explores the application of maximum-likelihood estimations for analyzing software reliability data and assessing software reliability. Furthermore, the paper presents instances of software reliability assessment using a tool that integrates various prominent SRGMs. These examples are based on observed test data derived from actual software projects. The drawback is the complexity of measuring software quality and reliability. Software systems are intricate and involve numerous interconnected components, making it challenging to develop comprehensive measurement techniques that capture all aspects of quality and reliability accurately. It is difficult to account for all potential failure scenarios and accurately predict the behavior of software in all possible circumstances.

V. Srivastava et al. [16] discovered a more robust software product source code for effective software product quality analysis. Software product quality analysis is challenging in the case of huge software applications. The profiles of resources are necessary for appropriate staffing, finance, and the advancement of an achievable project plan throughout the lifespan of a project, even while the needs for the framework may be functional but only stated at an extreme level. The size of the program may be calculated using past data and trends for the same project, opening up the possibility of an assessment approach. Software quality is now a key component in the intended attainment of overall human and commercial safety in the modern world, where processors are used in every conceivable area. Finding the software's quality attributes and turning them into calculable measurements will be crucial to the success of the final product. The mapping of program elements to these metric values illustrates details about the behavior and complexity of the framework. Here are some of the drawbacks associated with this technique are that it has limited scope, lack of context, overemphasis on quantitative metrics, sensitivity to programming languages and paradigms, the complexity of metric selection and interpretation, and inability to capture dynamic aspects.

E. Belachew [17] discussed the significance of software metrics in software engineering and emphasizes the importance of accurate measurement in this field. As software becomes more complex, manually inspecting it becomes challenging. Software engineers are concerned about software quality and seek ways to measure and improve it. The objective of the proposed research is to assess as well as analyze software metrics used for measuring software products and processes. The researcher collected literature from electronic databases since 2008 to gain a comprehensive understanding of software metrics. The study concludes that software quality can be measured by assessing how well the software aligns with its design, considering variables such as exactness, quality of product, scalability, fullness, and absence of bugs. Since quality standards vary across organizations, using software metrics and common tools can reduce subjectivity when evaluating software quality. The study's main contributions include providing an overview of software metrics and conducting a critical study of the key metrics found in the literature. On the other side, there are some potential limitations and drawbacks associated with this approach which are interpretation ambiguity, incomplete representation, metric selection, and validity, time and effort requirements, lack of causality, etc.

I. Khan et al. in [18], discussed that the field of Software Quality Analysis (SQA) is promptly mounting in the field of software product engineering, as it is aimed to offer more robust solutions for real-world applications. SQA involves a formal process utilized to assess, document, as well as ensure software application quality at all stages of the Software Development Life Cycle (SDLC). This research focuses on identifying and understanding various factors that can impact the quality of software product development. The study investigates the relationships between these factors and the different phases of the SDLC. Additionally, the research introduces a new quality factor called testability to the existing set of quality factors in system design and analysis. The anticipated outcomes of this suggested solution emphasize the significance of testability, especially during the

systems analysis and design stage of software development. One drawback of this process is its narrow focus, which may limit the understanding of software quality improvement. While testability is undeniably crucial in software development, it represents only a single facet of the broader concept of software quality. By primarily emphasizing testability, other vital factors that contribute to software quality, such as maintainability, scalability, performance, security, and usability may be overlooked.

R. Jamwal et al. [19] explored the increasing importance of software quality in today's marketplace and how it differs for producers and users of software. Software users view software as a tool to support their specific business sector, while quality is seen as a combination of various characteristics. These characteristics are often represented in models that depict their relationships, helping to highlight what people consider important in terms of quality. Different organizations adopt different quality models based on their requirements. This study examines various concepts of software quality appearances and provides a relative analysis of different software quality models used by different organizations.

3. METHODOLOGY

3.1. *Dataset:*

The researcher utilized the PROMISE Repository dataset in this study. Specifically, the NASA Metrics Data Program (MDP) provided the NASA PROMISE datasets, which include software metrics. Before training the ensemble model, multiple preprocessing tasks such as data normalization and transformation were conducted. These preprocessing operations involved addressing missing values and outliers, as well as selecting suitable feature variables for the input model. Subsequently, the proposed model was trained to forecast software flaws. Table 1 illustrates the datasets used for the experiment.

Table 1: ***Illustrated the Datasets Used for Experiment.***

Sr. No.	Dataset Name	Total Attributes	Total Instances	Fault Percentage
1.	CM1	24	500	8.98
2.	MC1	37	9464	1.1
3.	MW1	36	400	6.95
4.	PC1	24	1111	7.1
5.	MC2	43	163	30.90

3.2. *Design:*

In this study, a new design architecture is proposed which is mentioned in Figure 1. The developed EEM model aims to empirically assess and confirm the effectiveness of the suggested methods. The performance of this EEM model is evaluated on five NASA PROMISE repository datasets. The PROMISE repository dataset is a collection of software engineering datasets that are publicly available and widely used for research purposes. PROMISE stands for "Project Repository for Object-Oriented Software Engineering," and it aims to provide a comprehensive and diverse set of datasets related to software engineering tasks.

This entire dataset is initially preprocessed for training and testing purposes. Preprocessing refers to the techniques applied to raw data before it is used for analysis tasks. It involves transforming the data into a suitable format and preparing it for further analysis or modeling. Data

preprocessing is crucial as it helps to improve the quality of the data like handling missing values outlier detection and treatment feature scaling or normalization, feature selection or extraction, handling categorical variables, addressing issues or inconsistencies, and enhancing the performance of downstream tasks. In the next stage, preprocessed data are gone under hierarchical clustering which is a data analysis technique that groups similar data points into clusters in a hierarchical manner, forming a dendrogram a kind of tree-like structure. Each data point begins as its cluster and is eventually combined with others using a bottom-up technique based on their similarities, creating a hierarchy of clusters. Furthermore, the K-Fold Cross Validation is opted for datasets splitting into separate train as well as test groups.

The K-Fold Cross Validation method divides the dataset into two groups i.e., 80% data are train data, and the rest 20% data are test data. The train data in an ensemble model pertains to the dataset utilized for training the individual base models in the ensemble. It encompasses input samples along with their associated target values or labels. Each base model is trained separately using a subset of the training data, and their predictions are amalgamated to generate a final prediction. The testing data in an ensemble model refers to a separate dataset that is used to appraise the performance and generalization of the ensemble model. It consists of input samples, similar to the training data, but without the corresponding target values or labels. The testing data is used to determine how effectively the ensemble model can categorize or forecast brand-new data. The predictions of the individual base models are combined in the ensemble model to make predictions on the testing data and evaluate its accuracy and effectiveness. Furthermore, multiple algorithms which consist of Decision Tree (DT), AdaBoost, Random Forest (RF), and Naive Bayes (NB) are used with the help of a test model and pre-trained model to compute the performance. In the other direction, the Enhanced Ensemble Model which is the combination of the Extreme Learning Machine (ELM), K-Nearest Neighbors (KNN), and Support Vector Machine (SVM), is performed to compute the performance with the help of the test model and pre-trained model.

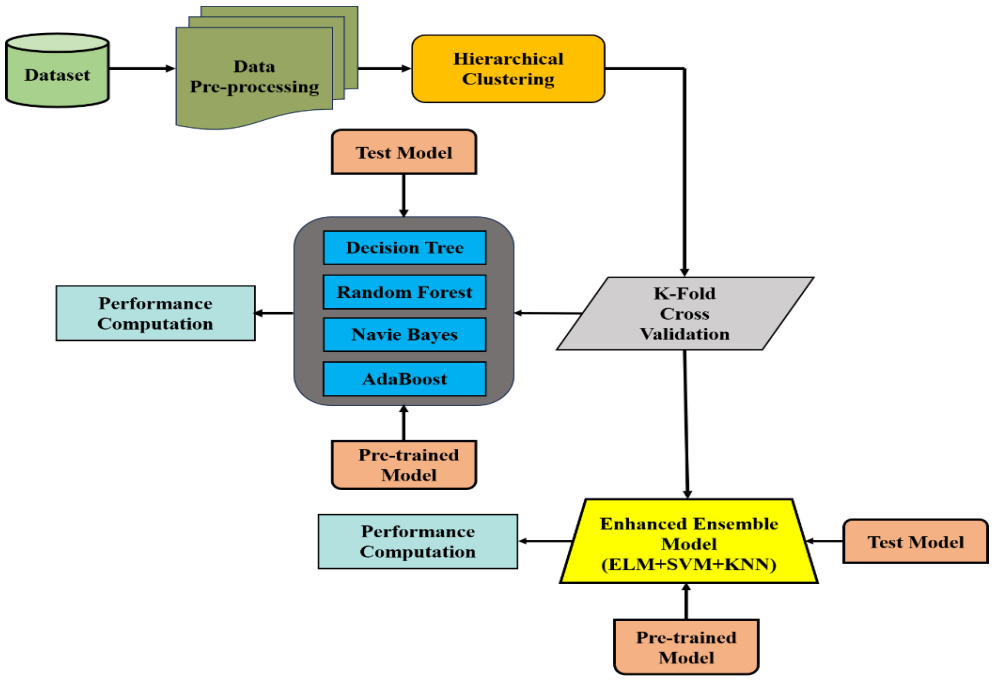

Figure 1: *Design Architecture of Proposed Model.*

3.3. *System Configuration:*

This section introduces the system configuration which is used in this research. For performing this research, there is utilized the latest system configuration. Table 2 represents the computer configuration and software details.

Table 2: ***Represents the System Configuration.***

Sr. No.	System Configuration	Details
1.	System RAM	8 GB
2.	Operating System	Windows 11
3.	Processor	13th Generation intel core i5
4.	Optimizer	Adam v1.3
5.	Programming Language	Python v3.11.0

3.4. *Performance Matrix:*

The four distinct performance measurement metrics, accuracy, recall, precision, and F1-score, are described in this section. These matrices are utilized to evaluate the effectiveness of this suggested EEM along with other algorithms on multiple NASA repository datasets.

3.4.1. *Precision*:

Precision is a metric used in ensemble learning machines to evaluate the accuracy of the ensemble model's successful predictions. It determines the proportion of true positives cases that were properly identified as positive to the total of true positives as well as false positives cases that were identified as positive. Precision aids in assessing the ensemble model's accuracy in locating positive examples. To fully comprehend the performance of the ensemble model in classification tasks must be taken into account in addition to other performance indicators.

$$\text{Precision}=TP/(FP+TP) \quad (1)$$

3.4.2. *Recall:*

The performance of an ensemble model to identify every positive case is measured by the recall metric. It aims to reduce false negatives cases in which a positive event is arbitrarily labeled as negative. The ratio of true positives (positive cases that were properly identified) to the total of true positives as well as false negatives is used to determine recall.

$$\text{Recall}=TP/(FN+TP) \quad (2)$$

3.4.3. *F1-Score:*

The harmonic mean of precision as well as recall that measures the proportion of positively predicted instances out of all positively predicted instances and measures the proportion of positively predicted instances out of all positively actualized instances, which is used to compute the F1-score. The F1-score is appropriate for unbalanced datasets where the distribution of classes is skewed since it takes the harmonic mean, which equalizes accuracy and recall.

$$\text{F1-Score}=(2\times\text{Recall}\times\text{Precision})/(\text{Recall}+\text{Precision}) \quad (3)$$

3.4.4. *Accuracy:*

Accuracy matrix refers to the measurement utilized for determining which model is better to

identify the relationships as well as patterns amongst the variables in a dataset rooted on the input, or training datasets.

$$Accuracy=(TN+TP)/(TN+TP+FN+FP) \quad (4)$$

4.RESULT AND DISCUSSION.

In this research, a novel EEM model is developed that is robust and performs the verification as well as validation in the software product development process. This model analyzes and improves selective parameters i.e., functional suitability along with software maintainability. The EEM model obtains maximum accuracy on diverse datasets of the NASA repository and performs better in comparison to other algorithms such as Decision Trees and many others for validation and verification of the software products. In addition to this, the proposed enhanced ensemble model analyzes the different selective parameter that is maintainability and functional suitability in a more effective manner. The analysis of software quality is done using an improved ensemble model based on selective these parameters. The EEM model precisely assesses the software system's quality and function suitability to improve the effectiveness of SDLC factors related to software quality, such as code complexity, maintainability, and performance. The EEM integrates ELM, SVM, and KNN to offer a thorough assessment of software quality. According to the analysis, this method provides pragmatic outcomes to identify possible problems or opportunities for software product improvement as well as improve software development processes.

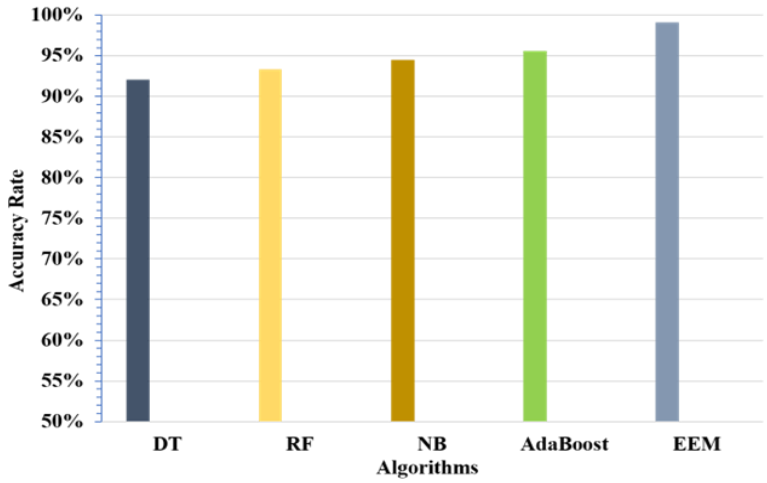

Figure 2*: Graphical Representation of Accuracy Rate.*

The graphical representation of different algorithms accuracy is shown in Figure 2. The accuracy rate has been computed and compared with the EEM approach. In this research, a performance comparison is demonstrated by implementing Decision Tree, Random Forest, Naive Bayes, and AdaBoost and compared with EEM. It is shown that the accuracy rate of Decision Tree is 92.8%, the accuracy measured on Random Forest is 93.35%, the accuracy on Naive Bayes is 94.50% and AdaBoost obtains an accuracy of 95.60%. However, the EEM obtains a maximum accuracy rate which is 99.13%. Hence, EEM is more efficient as compared to all the algorithms and offers more accurate outcomes on diverse datasets.

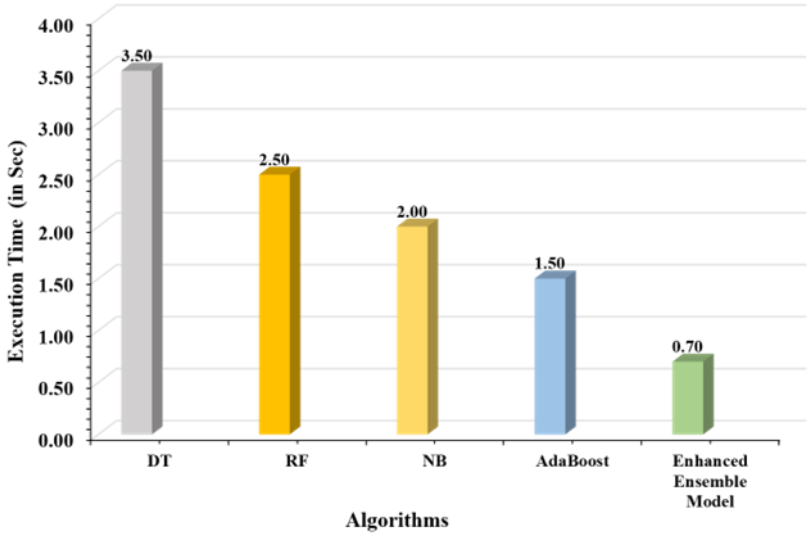

Figure 3: *Illustrates graphical representation of algorithms and their time of execution.*

Figure 3 illustrates a graphical representation of algorithms and their time of execution. While implementing the time taken by the Decision Tree, Random Forest, Navie Baye, and AdaBoost algorithm was computed at 3.50s, 2.50s, 2.00s, 1.50s, and 0.70s, respectively. The proposed EEM approach is executed with the lowest time i.e., 0.70s only. This demonstrates that this EEM is less complex in terms of time complexity and handles large data effectively in training and testing of the software product or applications.

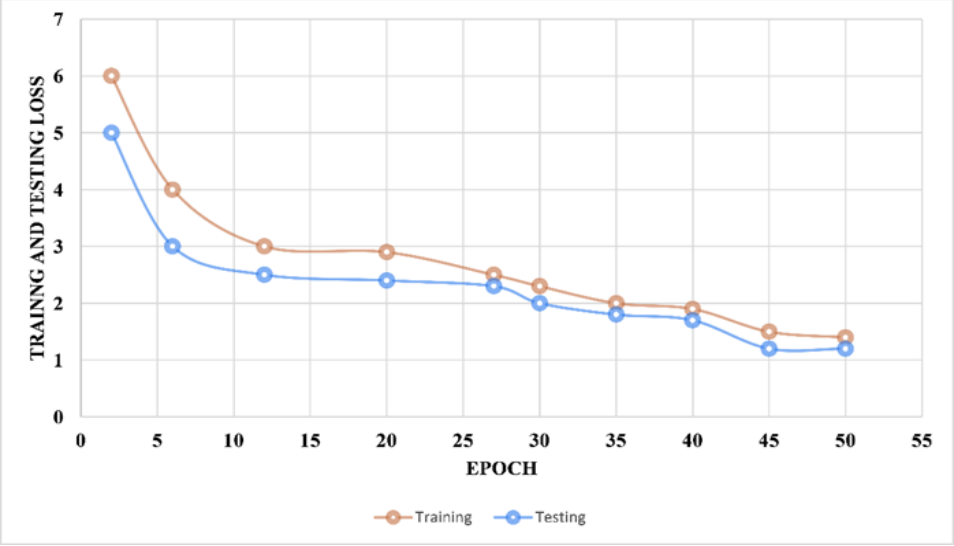

Figure 4: *Shows training as well as testing loss comparison.*

Figure 4 shows training as well as testing loss comparison of the proposed EEM approach. The computed training loss on epochs 2, 6, 12, 20, 27, 30, 35, 40, 45, and 50 is 6%, 4%, 3%, 2.9%, 2.5%, 2.3%, 2%, 1.9%, 1.5%, and 1.4%. The computed testing loss on epochs 2, 6, 12, 20, 27, 30, 35, 40, 45, and 50 is 5%, 3%, 2.5%, 2.4%, 2.3%, 2%, 1.8%, 1.7%, 1.2%, and 1.2%. Nevertheless, the proposed EEM approach performs well, and minimal train and test loss are seen.

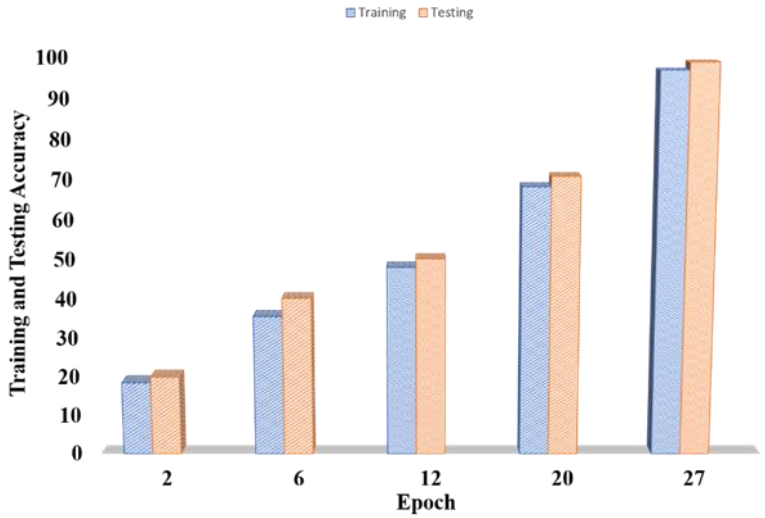

Figure 5: *Shows training as well as testing accuracy comparison.*

The training and testing accuracy of all algorithms along with the EEM approach are displayed in Figure 5. The training accuracy on epochs 2, 6, 12, 20, and 27 are 18.5%, 35.6%, 48.12%, 68.34%, and 97.01%. However, the test accuracy is found 19.98%, 40.22%, 50.25%, 70.91%, and 98.95%, respectively. It is shown that EEM achieves optimal accuracy level and is more suitable for software product quality analysis for diverse selective parameters such as functional suitability of the software product and maintainability. In addition to this, the EEM archives good accuracy for the software products and applications validation and verification.

Table 3: *Illustrates the Average Computed Accuracy of Different Algorithms.*

S. No.	Dataset Name	DT	AdaBoost	NB	RF	EEM
1.	CM1	0.89	0.88	0.90	0.91	0.98
2.	KC1	0.90	0.87	0.89	0.93	0.97
3.	MW1	0.91	0.92	0.91	0.94	0.99
4.	PC1	0.86	0.91	0.93	0.92	0.98
5.	MC2	0.88	0.89	0.89	0.95	0.99
Average		0.88	0.89	0.90	0.93	0.99

Table 3 illustrates the average computed accuracy of different algorithms along with EEM. The EEM and diverse algorithms are validated utilizing the NASA repository's five datasets namely, CM1, KC1, MW1, PC1, and MC2, respectively. The obtain accuracy using the decision tree algorithm on CM1, KC1, MW1, PC1, and MC2 are 0.89%, 0.90%, 0.91%, 0.86%, and 0.88%, respectively. The obtain accuracy using the AdaBoost algorithm on CM1, KC1, MW1, PC1, and MC2 are 0.88%, 0.87%, 0.92%, 0.91%, and 0.89%, respectively. The accuracy received using the NB algorithm on CM1, KC1, MW1, PC1, and MC2 are 0.90%, 0.89%, 0.91%, 0.93%, and 0.89%,

respectively. The accuracy received using the RF algorithm on CM1, KC1, MW1, PC1, and MC2 are 0.91%, 0.93%, 0.94%, 0.92%, and 0.95% respectively. The obtained accuracy using the EEM algorithm on CM1, KC1, MW1, PC1, and MC2 are 0.98%, 0.97%, 0.99%, 0.98%, and 0.99% respectively. The average accuracy on DT, AdaBoost, NB, RF, and EEM is found 0.88%%, 0.89%, 0.90%, 0.93%, and 0.99%, respectively.

5.CONCLUSION

Software quality analysis is becoming a very challenging and time-consuming process in modern software product and application development. There are developed multiple software testing approaches for effective analysis of the prediction of the faults in applications. However, these testing methods are inconsistent to determine the software product quality, especially in the developing phase, and consume a huge time for the developer in the unit testing process to determine the functional suitability and validation and verification of the product effectively. Therefore, this work presents an EEM approach that is based on the SVM, ELM, and KNN for software product and application quality analysis in less time. The obtained results are recorded and compared with the other algorithms i.e., DT, AdaBoost, NB, and RF for validation of the EEM. To validate the proposed EEM approach, there has been used the five diverse datasets of the PROMISE repository of NASA which includes the CM1, KC1, MW1, PC1, and MC2, respectively. It is found that the average accuracy on Decision Tree, Random Forest, Naive Bayes, AdaBoost, and EEM is 92.08%, 93.35%, 94.50%, 95.60%, and 99.14%, respectively.

References

[1] S. Alyahya, “Crowdsourced software testing: A systematic literature review,” Information and Software Technology. 2020. doi: 10.1016/j.infsof.2020.106363.

[2] D. Manikavelan and R. Ponnusamy, “Software quality analysis based on cost and error using fuzzy combined COCOMO model,” J. Ambient Intell. Humaniz. Comput., 2020, doi: 10.1007/s12652-020-01783-9.

[3] E. Ronchieri, M. Canaparo, and D. Salomoni, “A software quality model by using discriminant analysis predictive technique,” J. Integr. Des. Process Sci., 2014, doi: 10.3233/jid-2014-0016.

[4] OM Melkozerova “Software performance testing,” Bull. V.N. Karazin Kharkiv Natl. Univ. Ser. Mathematical Model. Inf. Technol. Autom. Control Syst., 2020, doi: 10.26565/2304-6201-2020-45-07.

[5] D. El-Masri, F. Petrillo, Y. G. Guéhéneuc, A. Hamou-Lhadj, and A. Bouziane, “A systematic literature review on automated log abstraction techniques,” Inf. Softw. Technol., 2020, doi: 10.1016/j.infsof.2020.106276.

[6] P. Temple, M. Acher, and J. M. Jezequel, “Empirical Assessment of Multimorphic Testing,” IEEE Trans. Softw. Eng., 2021, doi: 10.1109/TSE.2019.2926971.

[7] I. Atoum et al., “Challenges of Software Requirements Quality Assurance and Validation: A Systematic Literature Review,” IEEE Access. 2021. doi: 10.1109/ACCESS.2021.3117989.

[8] M. Varela-González, H. González-Jorge, B. Riveiro, and P. Arias, “Performance testing of LiDAR exploitation software,” Comput. Geosci., 2013, doi: 10.1016/j.cageo.2012.12.001.

[9] Z. Ding, J. Chen, and W. Shang, “Towards the use of the readily available tests from the release pipeline as performance tests,” in Proceedings - International Conference on Software Engineering, 2020. doi: 10.1145/3377811.3380351.

[10] S. M. M. A. Jammel Mona, Radhwan Hussein Abdulzhraa Al-Sagheer, “Software Quality Assurance Models and Application to Defect Prediction Techniques,” 2022, [Online]. Available:

https://ijisae.org/index.php/IJISAE/article/view/2455/1038

[11] E. A. Rashid, rikanta B. Patnaik, and V. C. Bhattacherjee, "Machine Learning and Software Quality Prediction: As an Expert System," Int. J. Inf. Eng. Electron. Bus., vol. 6, no. 2, pp. 9–27, Apr. 2014, doi: 10.5815/ijieeb.2014.02.02.

[12] T. Sharma, A. Jatain, S. Bhaskar, and K. Pabreja, "Ensemble Machine Learning Paradigms in Software Defect Prediction," Procedia Comput. Sci., vol. 218, pp. 199–209, 2023, doi: 10.1016/j.procs.2023.01.002.

[13] K. Bashir, T. Li, C. W. Yohannese, and Y. Mahama, "Enhancing software defect prediction using supervised-learning based framework," in 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Nov. 2017, pp. 1–6. doi: 10.1109/ISKE.2017.8258790.

[14] M. S. Rawat and S. K. Dubey, "Software defect prediction models for quality improvement: A literature study," Int. J. Comput. Sci. Issues, 2012.

[15] S. Yamada, "Software quality/reliability measurement and assessment: software reliability growth models and data analysis," J. Inf. Process., 1991.

[16] V. K. L. Srivastava, "An efficient Software Source Code Metrics for Implementing for Software quality analysis," Int. J. Emerg. Trends Eng. Res., pp. 216–222, Sep. 2019, doi: 10.30534/ijeter/2019/01792019.

[17] E. B. Belachew, "Analysis of Software Quality Using Software Metrics," Int. J. Comput. Sci. Appl., vol. 8, no. 4/5, pp. 11–20, Oct. 2018, doi: 10.5121/ijcsa.2018.8502.

[18] I. Khan and A. K. K. Shinwari, "Testability as a Measure for Improving Software Quality in System Analysis and Design," Kardan J. Eng. Technol., Dec. 2019, doi: 10.31841/KJET.2021.5.

[19] R. S. Jamwal, D. Jamwal, and D. Padha, "Comparative Analysis of Different Software Quality Models," Comput. Nation Dev., 2009.