RES MILITARIS
Social Science Journal

Advanced ML Models for Accurate Software Quality Prediction in Agile and Waterfall Development

Venkatesh Artham[1], Chenagoni Nagaraju[1], Mounika Pasam[1]
[1]Department of Computer Science and Engineering

[1]Sree Dattha Group of Institutions, Sheriguda, Telangana.

ABSTRACT

In the rapidly evolving domain of software engineering, the imperative to deliver high-quality software products has never been more pressing. Traditional software quality assurance practices, which primarily rely on manual code reviews, extensive testing, and debugging processes, often encounter significant limitations. Conventional methodologies such as Waterfall and Agile, while structured, frequently struggle to predict and prevent defects in the early stages of the software development lifecycle. This inability to identify issues proactively contributes to increased costs, delayed project timelines, and diminished software reliability.

The escalating complexity of modern software systems, combined with tight project schedules and the ever-increasing demand for high-quality products, underscores the necessity for innovative approaches to software quality prediction. In this context, machine learning (ML) methods emerge as a promising solution, enabling teams to leverage historical data to uncover patterns and correlations that may not be apparent through traditional methods. By analyzing past project data, ML algorithms can identify key indicators of software quality and predict potential defects before they materialize, thus facilitating timely interventions.

This research aims to develop advanced machine learning models that enhance the accuracy of software quality estimation by focusing on relevant features derived from large, diverse datasets. By employing a data-driven approach, the study seeks to identify which factors most significantly influence software quality and how these can be quantified to improve predictive capabilities. Additionally, the research will explore various machine learning techniques, including supervised and unsupervised learning, to determine the most effective strategies for quality prediction.

Furthermore, this work aspires to bridge the gap between traditional software quality assurance methods and the evolving needs of contemporary software development environments. By integrating machine learning techniques into the quality assurance process, organizations can achieve greater automation, efficiency, and adaptability, ultimately leading to enhanced software quality and improved customer satisfaction.

Through this innovative approach, the research endeavors to not only enhance the software development process but also contribute to the broader field of software engineering by providing actionable insights and frameworks for future quality assurance practices.

Keywords: Software Quality, Machine Learning, Software Development, Quality Prediction, Software Engineering

INTRODUCTION

1.1 Overview

Software applications may contain defects, originating from requirements analysis, specification and other activities conducted in the software development. Therefore, software quality estimation is an activity needed at various stages. It may be used for planning the project-based quality assurance

practices and for benchmarking. In addition, the number of defects per unit is considered one of the most important factors that indicate the quality of the software. There are two directly comparable studies on software quality prediction using defect quantities in ISBGS dataset. In the first study, the two methods (MCLP and MCQP) were experimented with the dataset and the results were compared.

The quality level was classified according to: number of minor defect + 2*number of major defect + 4*number of extreme defect. The quality of level was to be either high or low. They used k-fold cross-validation technique to measure MCLP and MCQP's performance on the ISBSG database. Release 10 Dataset (released in January 2007) which contained 4,017 records and 106 attributes was used. After preprocessing, 374 records and 11 attributes remained in the dataset. In another study, the same data set was used again. The software belonged to high quality class if it fulfils the following requirements: the extreme defects exist or the number of major defects is more than 1 or the number of minor defects is more than 10. The rest are assumed to belong to low quality class. After preprocessing, 746 projects and 53 attributes remained in the dataset. They used C5.0, SVM and Neutral network for classification.

As an example to a more application oriented study Rashid et al. [5] used case based reasoning (CBR) for software quality estimation. CBR is a machine learning model which performs the learning process using the results of the previous experiments. Line of code, number of functions, difficulty level, and development type and programmers experience are entered and these attributes are used for estimation. The deviation is calculated by using Euclidian distance (ED) or The Manhattan distance (MD). If the error in estimation is less than 10% then the record is saved to the database. Number of inputs that can be obtained from the user is limited. Also, it is necessary to have close values in the database in order to estimating precise values.

In these studies, quality estimation was done by binary classification. We tried to improve these prediction models, taking into account the size in terms of function points and using 4-level classification. We have experimented with recent classification methods shown to be successful for other prediction tasks.

1.2 Problem Statement

The problem statement for employing an AI-driven approach to enhance software quality prediction and estimation accuracy is multifaceted, reflecting the complexities and challenges inherent in modern software development processes. At its core lies the imperative to address the limitations of traditional estimation techniques, which often rely on simplistic heuristics and fail to capture the nuanced interplay of factors influencing software quality and project outcomes.

One facet of this problem is the inherent uncertainty and variability in software development projects, stemming from diverse technical, organizational, and environmental factors. Traditional estimation methods struggle to account for this complexity, leading to inaccurate predictions and costly project overruns. By harnessing the power of advanced machine learning algorithms, AI-driven approaches seek to mitigate this uncertainty by leveraging historical project data to uncover patterns and relationships that elude human intuition alone.

Furthermore, the proliferation of software systems characterized by ever-increasing scale, complexity, and interconnectedness exacerbates the challenges of accurate estimation. As projects grow in size and scope, predicting quality metrics such as defect density, code churn, and development effort becomes increasingly daunting. AI-driven approaches offer a path forward by enabling the analysis of vast and heterogeneous datasets, allowing for the identification of subtle indicators of software quality and project risk.

Moreover, traditional estimation techniques often overlook non-technical factors such as team dynamics, communication patterns, and stakeholder expectations, which can exert a significant influence on project outcomes. AI-driven approaches provide a means to incorporate these soft factors into predictive models, thereby enriching estimation accuracy and enhancing the alignment between project plans and organizational objectives.

However, the adoption of AI-driven approaches for software quality prediction is not without its challenges. Data quality issues, such as incompleteness, inconsistency, and bias, pose significant obstacles to model development and deployment. Furthermore, ensuring the interpretability and transparency of AI-driven predictions is essential to foster trust and facilitate collaboration between data scientists, software engineers, and project stakeholders.

In summary, the problem statement for employing an AI-driven approach to improve software quality prediction and estimation accuracy revolves around the need to overcome the limitations of traditional techniques in the face of increasing project complexity, uncertainty, and non-technical influences. By harnessing the power of advanced machine learning algorithms and interdisciplinary collaboration, organizations can pave the way towards more reliable, informed, and successful software development practices.

1.4 Motivation

The research motivation behind employing AI-driven approaches for software quality prediction with the goal of improving estimation accuracy stems from the pressing need within the software engineering community to mitigate the inherent uncertainties and complexities associated with software development projects. Traditional estimation techniques often rely on simplistic models and subjective judgments, leading to suboptimal.

Furthermore, the research motivation extends to the pursuit of innovation and advancement in the field of software engineering. By pushing the boundaries of AI-driven techniques, researchers aim to develop novel methodologies and tools that not only improve estimation accuracy but also foster a deeper understanding of software quality attributes and their impact on project outcomes. Ultimately, the goal is to empower software development teams with the insights and tools needed to make informed decisions, mitigate risks, and deliver high-quality software products in a timely and efficient manner.

2. LITERATURE SURVEY

Software quality metrics in quality assurance to study the impact of external factors related to time:

Software quality assurance is a formal process for evaluating and documenting the quality of the work products during each stage of the software development lifecycle. The practice of applying software metrics to operational factors and to maintain factors is a complex task. Successful software quality assurance is highly dependent on software metrics. It needs linkage the software quality model and software metrics through quality factors in order to offer measure method for software quality assurance. The contributions of this paper build an appropriate method of Software quality metrics application in quality life cycle with software quality assurance. Design: The purpose approach defines some software metrics in the factors and discussed several software quality assurance model and some quality factors measure method. Methodology: This paper solves customer value evaluation problem are: Build a framework of combination of software quality criteria. Describes software metrics. Build Software quality metrics application in quality life cycle with software quality assurance. Results: From the appropriate method of Software quality metrics application in quality life cycle with software quality

assurance, each activity in the software life cycle, there is one or more QA quality measure metrics focus on ensuring the quality of the process and the resulting product. Future research is need to extend and improve the methodology to extend metrics that have been validated on one project, using our criteria, valid measures of quality on future software project.

Software defect prediction: do different classifiers find the same defects:

During the last 10 years, hundreds of different defect prediction models have been published. The performance of the classifiers used in these models is reported to be similar with models rarely performing above the predictive performance ceiling of about 80% recall. We investigate the individual defects that four classifiers predict and analyse the level of prediction uncertainty produced by these classifiers. We perform a sensitivity analysis to compare the performance of Random Forest, Naïve Bayes, RPart and SVM classifiers when predicting defects in NASA, open source and commercial datasets. The defect predictions that each classifier makes is captured in a confusion matrix and the prediction uncertainty of each classifier is compared. Despite similar predictive performance values for these four classifiers, each detects different sets of defects. Some classifiers are more consistent in predicting defects than others. Our results confirm that a unique subset of defects can be detected by specific classifiers. However, while some classifiers are consistent in the predictions they make, other classifiers vary in their predictions. Given our results, we conclude that classifier ensembles with decision-making strategies not based on majority voting are likely to perform best in defect prediction.

A Knowledge Discovery Case Study of Software Quality Prediction:

Software becomes more and more important in modern society. However, the quality of software is influenced by many un-trustworthy factors. This paper applies MCLP model on ISBSG database to predict the quality of software and reveal the relation between the quality and development attributes. The experimental result shows that the quality level of software can be well predicted by MCLP Model. Besides, several useful conclusions have been drawn from the experimental result.

Evidence-based software portfolio management:

In this paper we describe and evaluate a tool for Evidence-Based Software Portfolio Management (EBSPM) that we developed over time in close cooperation with software practitioners from The Netherlands and Belgium. Objectives: The goal of the EBSPM-tool is to measure, analyze, and benchmark the performance of interconnected sets of software projects in terms of size, cost, duration, and number of defects, in order to support innovation of a company's software delivery capability. The tool supports building and maintaining a research repository of finalized software projects from different companies, business domains, and delivery approaches. Method: The tool consists of two parts. First, a Research Repository, at this moment holding data of for now 490 finalized software projects, from four different companies. Second, a Performance Dashboard, built from a so-called Cost Duration Matrix. Results: We evaluated the tool by describing its use in two practical applications in case studies in industry. Conclusions: We show that the EBSPM-tool can be used successfully in an industrial context, especially regarding its benchmarking and visualization purposes.

3. PROPOSED METHD

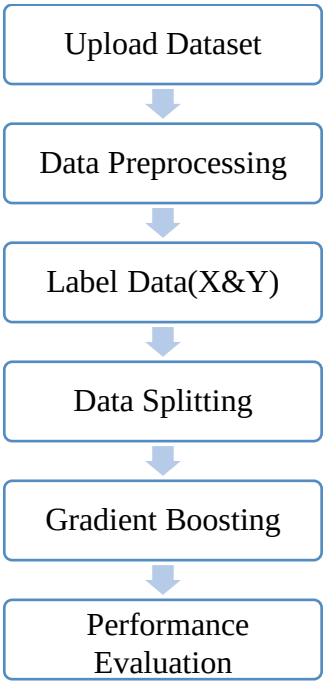

Figure 1: Block Diagram Proposed System architecture.

XG Boost Model

XGBoost is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "XGBoost is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the XGBoost takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

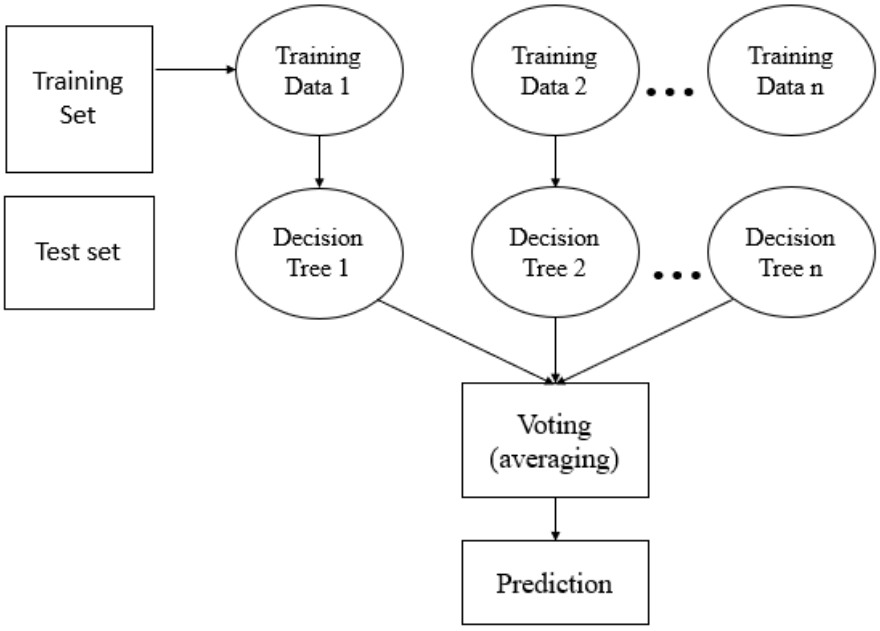

Fig. 2: XGBoost algorithm.

XGBoost, which stands for "Extreme Gradient Boosting," is a popular and powerful machine learning algorithm used for both classification and regression tasks. It is known for its high predictive accuracy and efficiency, and it has won numerous data science competitions and is widely used in industry and academia. Here are some key characteristics and concepts related to the XGBoost algorithm:

- **Gradient Boosting:** XGBoost is an ensemble learning method based on the gradient boosting framework. It builds a predictive model by combining the predictions of multiple weak learners (typically decision trees) into a single, stronger model.
- **Tree-based Models:** Decision trees are the weak learners used in XGBoost. These are shallow trees, often referred to as "stumps" or "shallow trees," which helps prevent overfitting.
- **Objective Function:** XGBoost uses a specific objective function that needs to be optimized during training. The objective function consists of two parts: a loss function that quantifies the error between predicted and actual values and a regularization term to control model complexity and prevent overfitting. The most common loss functions are for regression (e.g., Mean Squared Error) and classification (e.g., Log Loss).
- **Gradient Descent Optimization:** XGBoost optimizes the objective function using gradient descent. It calculates the gradients of the objective function with respect to the model's predictions and updates the model iteratively to minimize the loss.
- **Regularization:** XGBoost provides several regularization techniques, such as L1 (Lasso) and L2 (Ridge) regularization, to control overfitting. These regularization terms are added to the objective function.
- **Parallel and Distributed Computing:** XGBoost is designed to be highly efficient. It can take advantage of parallel processing and distributed computing to train models quickly, making it suitable for large datasets.
- **Handling Missing Data:** XGBoost has built-in capabilities to handle missing data without requiring imputation. It does this by finding the optimal split for missing values during tree construction.
- **Feature Importance:** XGBoost provides a way to measure the importance of each feature in the model. This can help in feature selection and understanding which features contribute the most to the predictions.
- **Early Stopping:** To prevent overfitting, XGBoost supports early stopping, which allows training to stop when the model's performance on a validation dataset starts to degrade.
- **Scalability:** XGBoost is versatile and can be applied to a wide range of machine learning tasks, including classification, regression, ranking, and more.
- **Python and R Libraries:** XGBoost is available through libraries in Python (e.g., **xgboost**) and R (e.g., **xgboost**), making it accessible and easy to use for data scientists and machine learning practitioners.

4 RESULTS AND DISCUSSION

Figure 1 illustrates the graphical user interface (GUI) of an AI-driven approach designed for software quality prediction. The interface appears user-friendly and intuitive, featuring various buttons and sections for seamless interaction with the system. Figure 2 showcases the functionality for uploading datasets. It provides users with a straightforward mechanism to input their data, facilitating further analysis and prediction tasks within the system. Figure 3 presents a graphical representation of dataset features, possibly in the form of histograms, bar charts, or scatter plots. This visualization aids users in gaining insights into the distribution and characteristics of the dataset. Figure 4 displays the uploaded

dataset within the GUI. It presents a tabular view of the data, allowing users to review and verify the information they have inputted. Figure 5 offers a graphical depiction of the values contained in the uploaded dataset. This visualization could help users understand the range, variability, and patterns present in the data.

Figure 6 demonstrates the preprocessing steps applied to the uploaded dataset. This include tasks such as handling missing values, encoding categorical variables, and scaling numerical features. Figure 7 showcases data visualization using a heatmap. Heatmaps are effective for displaying correlations between different features in the dataset, providing insights into potential relationships and dependencies.

Figure 1: GUI of AI Driven Approach for Software Quality Prediction

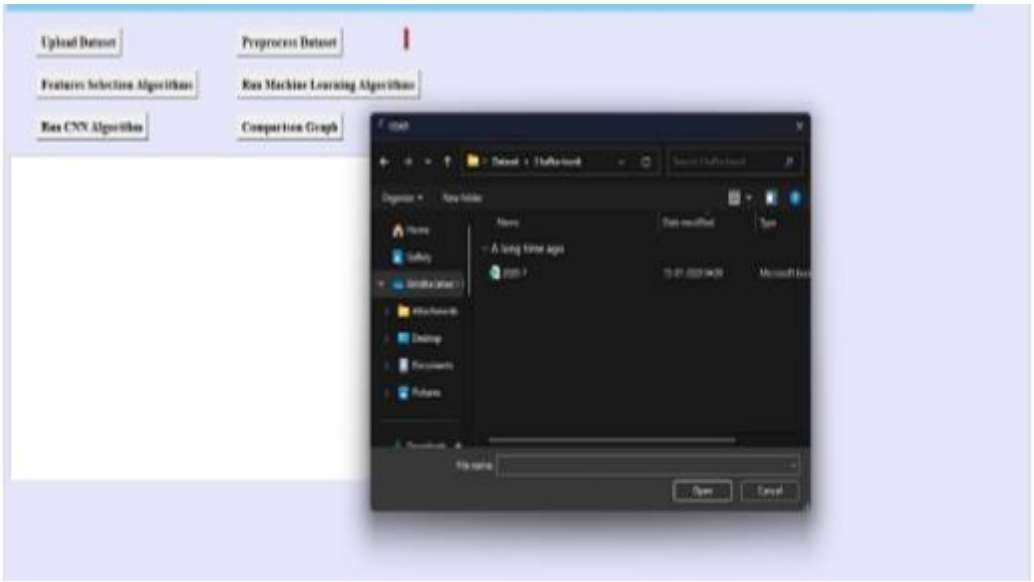

Figure 2: Presents the Uploading Dataset.

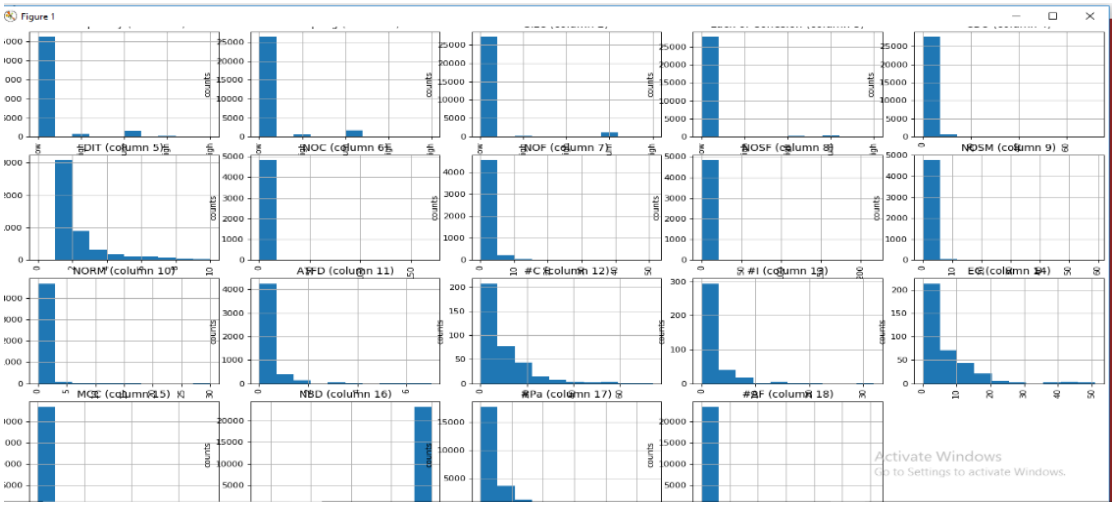

Figure 3: Graphical representation of Dataset Features.

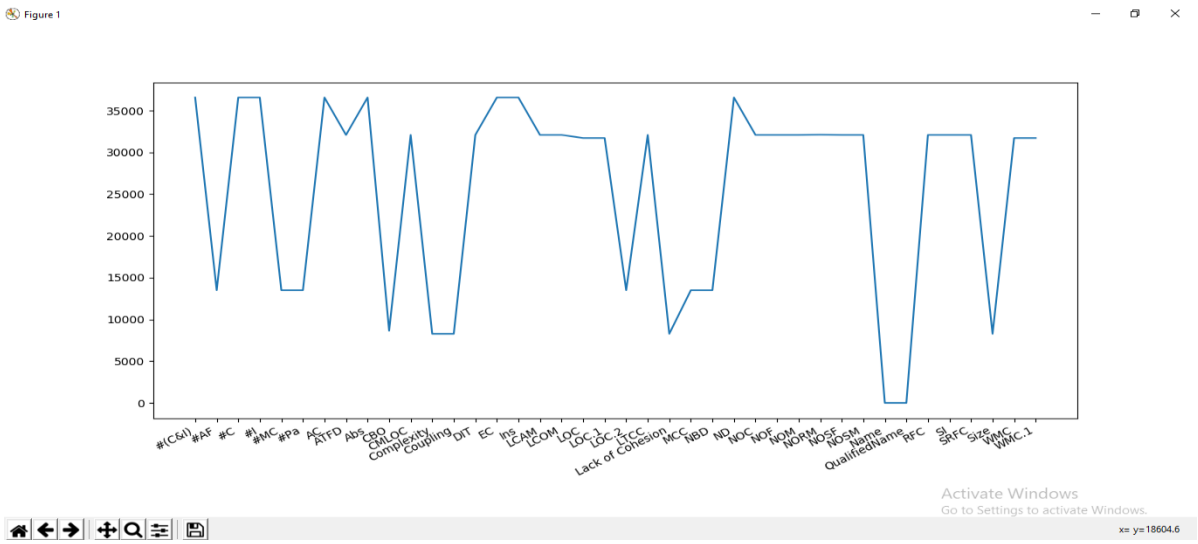

Figure 5: Shows the Graphical Values of Uploaded dataset.

Figure 6: Preprocessing the Uploaded dataset.

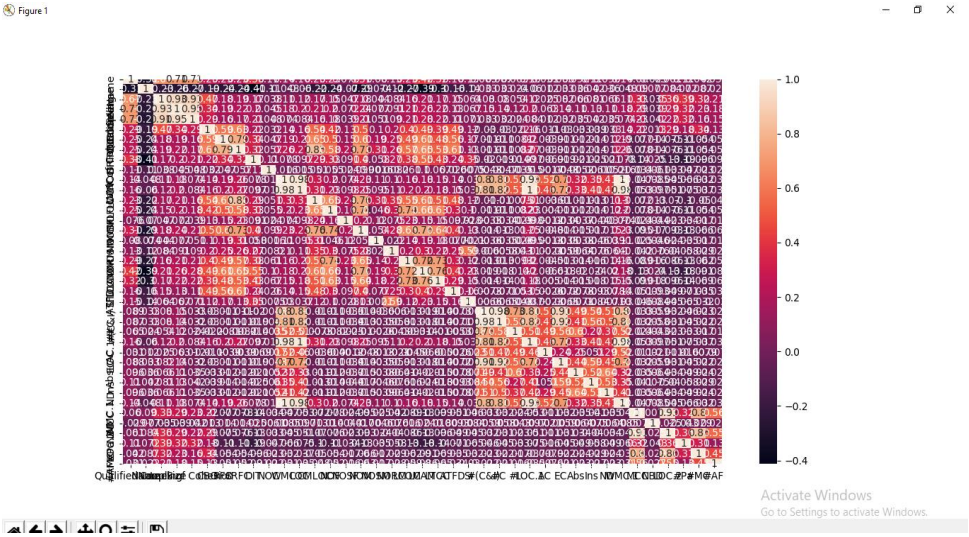

Figure 7: Data Visualization using Heatmap

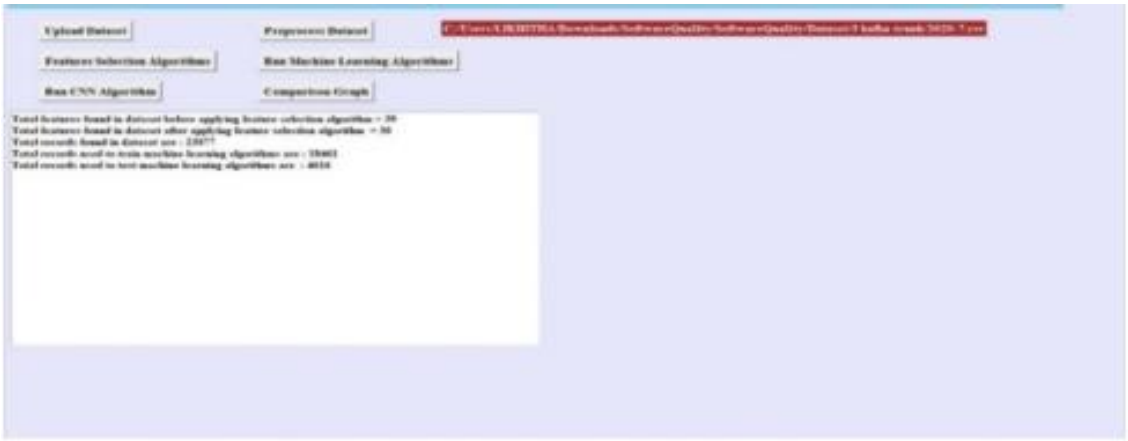

Figure 8: Display the Total Features.

Figure 9: Display the performance metrics of ML Algorithms

Figure 10: Displays CNN Algorithm Performance metrices.

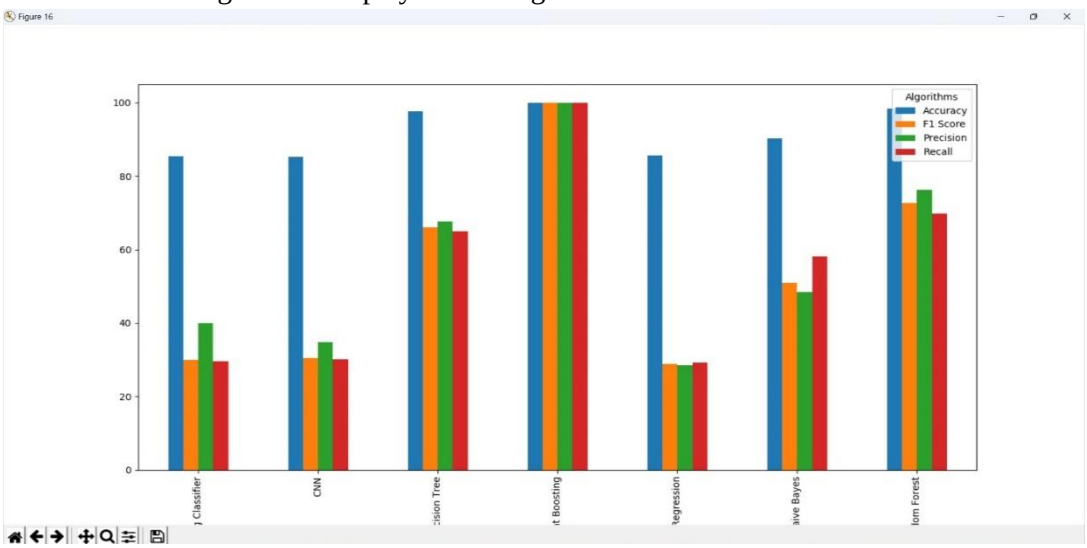

Fig 11: Graphical representation of Performance metrices

Figure 8 provides a summary of the total features present in the dataset. This information enables users to understand the dimensionality of the data and the complexity of the prediction task. Figure 9 exhibits the performance metrics of machine learning (ML) algorithms utilized in the system. This include metrics such as accuracy, precision, recall, and F1-score, allowing users to assess the effectiveness of different algorithms. Figure 10 focuses specifically on the performance metrics of a Convolutional Neural Network (CNN) algorithm. CNNs are commonly used in image recognition tasks, and this figure highlights the algorithm's accuracy and other relevant metrics. Fig 11 offers a graphical representation of the performance metrics, possibly in the form of bar charts or line graphs. Visualizing the metrics aids users in comparing the performance of different algorithms and making informed decisions.

5. CONCLUSION

Successful implementation of a software product entirely depends on the quality of the software developed. However, prediction of the quality of a software product prior to its implementation in real-world applications presents significant challenges to the software developer during the process of development. A limited spectrum of research in this area has been reported in the literature as of today. We have experimented with recent algorithms that support multi-class classification. The accuracies achieved by using these algorithms are impressive as compared to existing models. In comparison to previous directly comparable studies, acceptable level multiclass quality prediction could be achieved.

REFERENCES

[1] Vijay, T. John, D. M. G. Chand, and D. H. Done. "Software quality metrics in quality assurance to study the impact of external factors related to time." International Journal of Advanced Research in Computer Science and Software Engineering, 2017.
[2] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?." Software Quality Journal, 26(2), 2018, pp. 525-552.
[3] X. Wang, Y. Zhang, L. Zhang and Y. Shi, "A Knowledge Discovery Case Study of Software Quality Prediction: ISBSG Database," 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Toronto, ON, 2010, pp. 219-222.
[4] X. Wang, Y. Zhang, L. Zhang and Y. Shi, "A Knowledge Discovery Case Study of Software Quality Prediction Based on Classification Models: ISBSG Database," The 11th International Symposium on Knowledge Systems Sciences (KSS 2010), 2010
[5] E. Rashid, S. Patnaik, and V. Bhattacherjee, "Software quality estimation sing machine learning: Case-Based reasoning technique, " International Journal of Computer Applications, 2012
[6] www.isbsg.org
[7] https://goverdson.nl/
[8] H. Huijgens,"Evidence-based software portfolio management: a tool description and evaluation", 20th International Conference on Evaluation and Assessment in Software Engineering (EASE '16), 2016.