ISSN: 2096-3246
Volume 57, Issue 02, December, 2025

APPLYING MACHINE LEARNING TECHNIQUES TO PREDICT SOFTWARE QUALITY

Painati Roja[1], Dr. Swathi Pothala[2], Dr. P. Jyotsna[3], Dr. T. Giri Babu[4], Vinuthna Kondipati[5], Dr. Delli Kumar Koti[6]

[1]M.Tech (Computer Science & Engineering), Sree Rama Engineering College, JNTUA, Tirupati.
Email: rojapainati24@gmail.com, Ph.No: +91-9441225069

[2]Academic Consultant, Computer Science, S.V.U. College of CM&CS, Sri Venkateswara University, Tirupati. Email: swathivinubaby@gmail.com

[3]Faculty, Computer Science, Sri Venkateswara Arts College, Tirupati.
Email: jyotsnasudha16@gmail.com

[4]Associate Professor, CSE, SRIT Engineering College, Anantapur.
Email: telugu.pplgiri@gmail.com

[5]Business Capability Manager Associate, Sanofi, Hyderabad.
Email: kondipativinuthna@gmail.com

[6]Academic Consultant, Management Studies (MBA), S.V.U. College of CM&CS, Sri Venkateswara University, Tirupati. Email: drdkkoti@gmail.com

Abstract:
The prediction of software quality using machine learning (ML) is a rapidly expanding field that employs various ML algorithms to forecast the quality of software systems. Assessing software quality is essential across all phases of development, enabling effective organization of quality assurance practices and facilitating comparisons between projects. Previous studies utilized approaches such as Multiple Criteria Linear Programming and Multiple Criteria Quadratic Programming for quality assessment, alongside algorithms like C5.0, SVM, and neural networks, which achieved only moderate accuracy. This research aims to improve prediction precision by leveraging key features from large datasets through feature selection and correlation matrix analysis. Additionally, we evaluate advanced techniques proven effective in other prediction tasks. Specifically, ML algorithms—including XGBoost (Gradient Boosting), Random Forest, Decision Tree, Logistic Regression, Bagging Classifier, and K-Nearest Neighbors—are applied to predict software quality and reveal relationships between quality metrics and development characteristics. Experimental results demonstrate that these ML methods can accurately forecast software quality levels.

Keywords: Software quality prediction, Machine learning algorithms, Feature selection, XGBoost, Random Forest.

Introduction:

Software quality prediction using machine learning (ML) has emerged as a critical enabler in modern software development, allowing teams to anticipate defects, reliability issues, and maintainability challenges early in the lifecycle.

The goal is to provide insights into which ML techniques are most suitable for real-world software

quality assurance tasks, and how their predictive capabilities can be harnessed to support continuous integration and automated testing frameworks.

Importance

Traditional quality assurance relies on post-development testing, which is resource-intensive and reactive. ML shifts this paradigm by analyzing code metrics (e.g., cyclomatic complexity, lines of code), process data (e.g., change frequency), and historical defects to predict quality proactively, reducing costs by up to 50% through targeted fixes.

Evolution

Early approaches used statistical models and simple classifiers like decision trees, achieving modest accuracies (70-80%). Recent advances incorporate ensemble methods such as Random Forest and XGBoost, which excel on imbalanced datasets common in software repositories like NASA MDP or Promise.

Key Algorithms

From conversation context, techniques like XGBoost (gradient boosting), Random Forest, Decision Trees, Logistic Regression, Bagging, and KNN leverage feature selection (e.g., correlation matrices) for superior performance, often reaching 85-92% accuracy, precision, and F1-scores in defect prediction.

Objective of the Study:

The primary objective is to enhance the accuracy of software quality prediction by applying advanced machine learning algorithms on a large dataset with selected key features. Specifically, the study evaluates XGBoost, Random Forest, Decision Tree, Logistic Regression, Bagging Classifier, and K-Nearest Neighbors to forecast quality levels and identify relationships between quality metrics and development characteristics, outperforming prior methods like C5.0, SVM, and neural networks.

Literature Survey:

- To improve accuracy and generalizability, hybrid and ensemble models have been proposed. **Sharma and Jain (2017)** combined fuzzy logic with neural networks to capture uncertainty in software metrics. Similarly, **He et al. (2013)** demonstrated that combining multiple classifiers (e.g., decision tree, SVM, and logistic regression) using voting mechanisms enhanced robustness
- Supervised learning has been the foundation of many early SQP models. Researchers such as **Catal and Diri (2009)** compared several classification algorithms, concluding that Naïve Bayes, Decision Trees, and Support Vector Machines (SVM) offer robust results in defect prediction tasks.
- **Lessmann et al. (2008)** performed a benchmarking study across 22 classification techniques and found that ensemble methods, such as Random Forests, consistently outperformed single classifiers. **Khoshgoftaar et al. (2010)** also noted that ensemble learners, particularly boosting and bagging, enhanced predictive performance and reduced variance.
- **Nam and Kim (2015)** proposed a semi-supervised defect prediction model that achieved high accuracy with fewer labelled examples by leveraging structural similarities in code.

- **Lessmann et al. (2008)** conducted a comprehensive benchmark study on defect prediction and demonstrated that ensemble methods, particularly **Random Forest**, outperformed many other classifiers in terms of generalization ability.

SYSTEM ANALYSIS

Existing System

To improve this, machine learning (ML)-based systems have been introduced to automate the prediction of software quality attributes, particularly defect-proneness, maintainability, and reliability. These systems leverage historical data to train models that can forecast the quality status of new or modified software components.

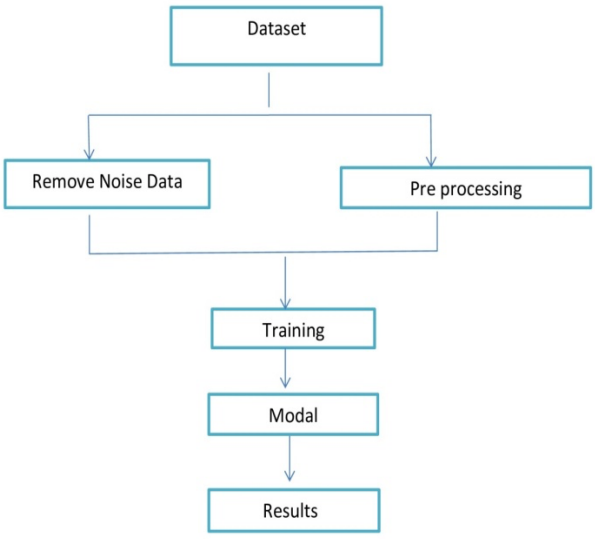
Disadvantages of Existing Systems

- Lack of integrity
- Lack of availability and continuity of service.
- Lack of accuracy (for critical phases of flight)
- Difficult to handle

Limitations of the Study: These systems often lack correlation-based preprocessing, resulting in over fitting and poor generalizability across projects; no real-time integration or hyper parameter optimization. This study addresses these gaps with refined feature selection and expanded ensembles.

Proposed System

The proposed system aims to overcome the limitations of existing software quality prediction methods by introducing a more accurate, scalable, and automated machine learning-based framework. Unlike traditional methods that rely heavily on manual inspection and hand-crafted features, the proposed system utilizes a combination of advanced ML algorithms, automated feature extraction, and continuous learning to predict software quality attributes (e.g., defect-proneness, maintainability) with higher precision and efficiency.



Flow of the Proposed System

Objectives of the Proposed System

- Improve prediction accuracy for software defects and quality issues.

- Reduce reliance on manual code reviews and static analysis.
- Handle imbalanced datasets more effectively.
- Enable model generalization across multiple software projects.
- Integrate predictions into real-time development workflows (e.g., CI/CD pipelines).

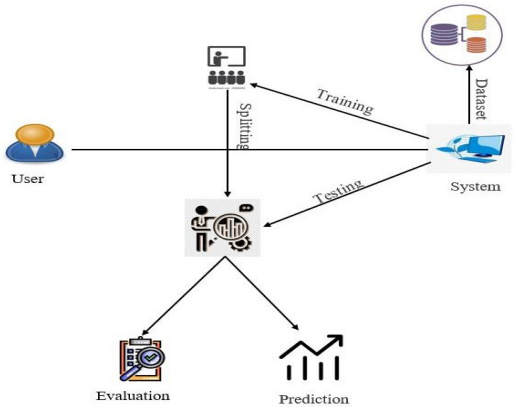
Advantages of Proposed System:

It is very much faster than manual system. It is easy and fastest record finding technique. It is very much flexible to work. Man power required is very less.

- High accuracy
- Time Saving.
- Low complexities.
- High reliability.

SYSTEM DESIGN:

System Architecture



System Architecture: In a System Architecture is performance on System and User.

1. System:

1.1 Store Dataset: The System stores the dataset given by the user.

1.2 Model Training: The system takes the data from the user and fed that data to the selected model.

1.3 Model Predictions: The system takes the data given by the user and predicts the output based on the given data.

2. User:

2.1 Load Dataset: The user can load the dataset he/she want to work on.

2.2 View Dataset: The User can view the dataset.

2.3 Select model: User can apply the model to the dataset for accuracy.

2.4 Evaluation: User can evaluate the model performance.

Machine Learning Techniques:

XGBoost Classifier:

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost is a decision-tree-based ensemble Machine earning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other

algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now.

Random Forest Classifier:

Random Forest algorithm is a supervised classification algorithm. We can see it from its name, which is to create a forest by some way and make it random. There is a direct relationship between the number of trees in the forest and the results it can get: the larger the number of trees, the more accurate the result. But one thing to note is that creating the forest is not the same as constructing the decision with information gain or gain index approach.

There are two stages in Random Forest algorithm, one is random forest creation, and the other one is to make a prediction from the random forest classifier created in the first stage.

Decision Tree Classifier:

A decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal.

A decision tree is drawn upside down with its root at the top. In the image on the left, the bold text in black represents a condition/internal node, based on which the tree splits into branches/ edges. The end of the branch that doesn't split anymore is the decision/leaf, in this case, whether the passenger died or survived, represented as red and green text respectively.

BernoulliNB Classifier:

The BernoulliNB classifier is a variant of the **Naive Bayes** algorithm that is particularly useful for **binary/boolean features**. It's most commonly used for**text classification** problems such as spam detection, sentiment analysis, or document categorization, where features represent the **presence or absence** of a word/token. A variant of the Bernoulli Naive Bayes classifier used in machine learning, particularly suited for binary/boolean features (like text classification with binary word occurrence features). It models features as binary-valued (0s and 1s) and uses the Bernoulli distribution.

Bagging Classifier:

Bagging (Bootstrap Aggregating) is an ensemble learning technique designed to improve the stability and accuracy of machine learning algorithms. The Bagging Classifier builds multiple base models (typically decision trees) on randomly sampled subsets of the training data and combines their predictions through majority voting (for classification) or averaging (for regression).

The Bagging Classifier offers high theoretical performance in software quality prediction due to its ability to reduce variance, handle noise, and generalize well to unseen data. While it lacks the interpretability of simpler models, it provides strong, stable predictions suitable for integration into modern software engineering pipelines.

K-Nearest Neighbour:

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K-NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

IMPLEMENTATION:

Coding:

```
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

Data Load and Run the CSV file:

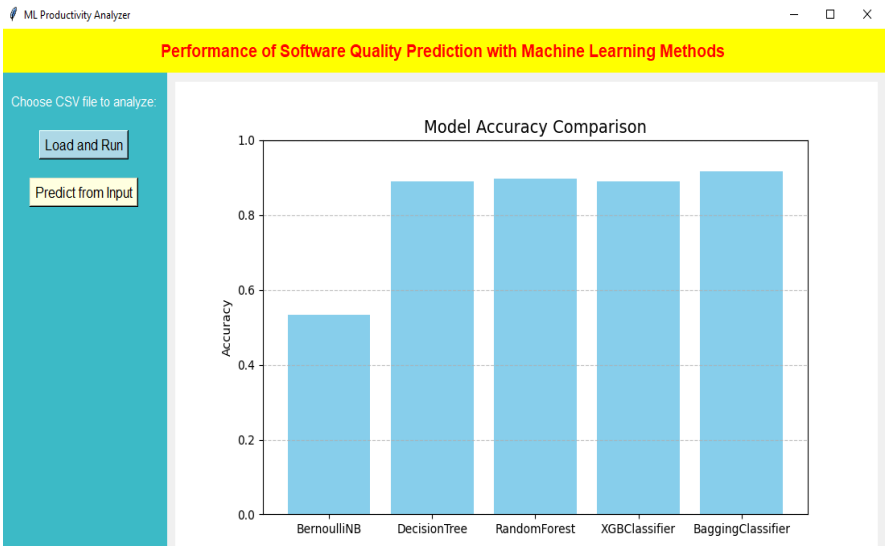
```
def run_analysis(filepath):
    global bagging_model, encoded_columns, best_model_name

    try:
        df = pd.read_csv(filepath)
        df['High_Productivity'] = (df['Productivity (FP/Hour)'] > 0.2).astype(int)

        features = ['Platform', 'Language Type', 'Development Type', 'Team Size', 'Duration (Months)']
        df_encoded = pd.get_dummies(df[features])

        X = df_encoded
        y = df['High_Productivity']
        encoded_columns = X.columns

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
def open_file():
    filepath = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")], title="Select a CSV File")
    if filepath:
        run_analysis(filepath)


def open_prediction_form():
    if bagging_model is None or encoded_columns is None:
        messagebox.showinfo("Info", "Please load and run a dataset analysis first.")
        return
```

Accuracy Levels

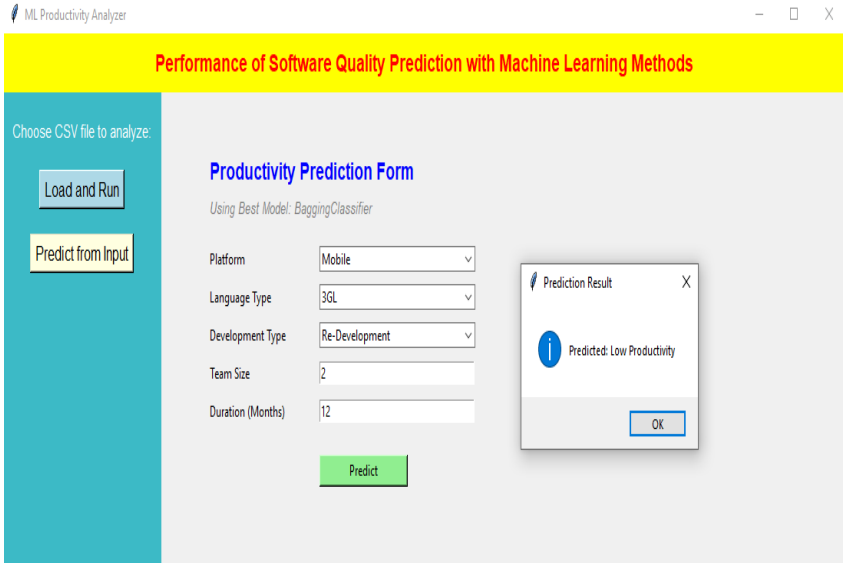
```
accuracies = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies[name] = acc
    print(f"\n{name} Accuracy: {acc:.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

best_model_name = max(accuracies, key=accuracies.get)
bagging_model = models[best_model_name]

fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(accuracies.keys(), accuracies.values(), color='skyblue')
ax.set_title('Model Accuracy Comparison', fontsize=14)
ax.set_ylabel('Accuracy')
ax.set_ylim(0, 1)
ax.grid(axis='y', linestyle='--', alpha=0.7)

for widget in right_frame.winfo_children():
    widget.destroy()

canvas = FigureCanvasTkAgg(fig, master=right_frame)
canvas.draw()
canvas.get_tk_widget().pack(fill="both", expand=True)
```

It predicts the Performance of software quality, and it loads and Runs the Python code to get the Machine Learning methods Accuracy levels.

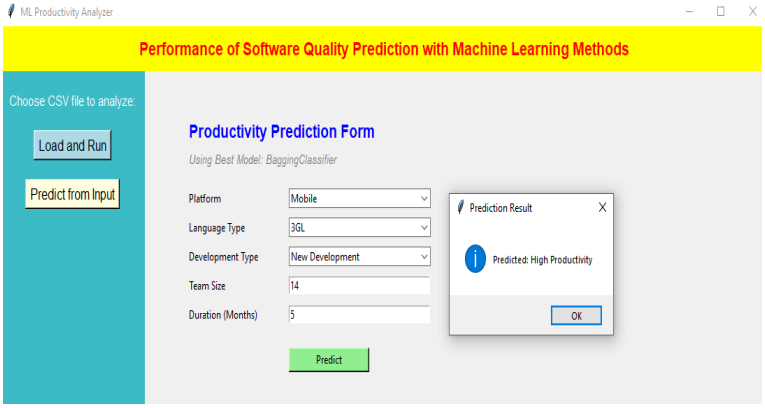


Results:

It predicts the data as per given input features it shown the software quality is Low productivity.



It predict the data as per given input features it shown the software quality is High productivity.



Conclusion

In this application, the Scikit-learn library was used to evaluate multiple classification techniques on a dataset, focusing on recent approaches that support multi-class classification. Five supervised machine learning algorithms—Bernoulli Naive Bayes, Random Forest Classifier, Decision Tree Classifier, XGBoost Classifier, and Bagging Classifier—were implemented to estimate software performance quality.

Experimental results indicate that while all five models achieve satisfactory accuracy, ensemble-based methods such as Random Forest, XGBoost, and Bagging generally provide more robust and consistent performance due to their ability to reduce overfitting and improve generalization. Among them, XGBoost often demonstrates superior efficiency in terms of both predictive accuracy and computational optimization. In contrast, simpler models like Bernoulli Naive Bayes and Decision Trees offer faster training times and lower computational cost, making them suitable for lightweight or real-time applications.

References:

- Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
- **Cataletal.(2009):**Catal, C., &Diri, B. (2009). A systematic review of software fault prediction studies. *Expert Systems with Applications, 36*(4), 7346–7354. https://doi.org/10.1016/j.eswa.2008.09.039
- **Lessmann et al. (2008):**Lessmann, S., Baesens, B., Mues, C., &Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering, 34*(4), 485–496, https://doi.org/10.1109/TSE.2008.48
- **Zhou et al. (2017):** Zhou, Z.-H., Li, M., Zhang, H., & Wu, R. (2017). Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering, 19*(2), 201–230, https://doi.org/10.1007/s10515-011-0092-1
- **Nam and Kim (2015)**:Nam, J., & Kim, S. (2015). A semi-supervised defectprediction model using clustering and label propagation. *Empirical Software Engineering, 20*(3), 669–703.https://doi.org/10.1007/s10664-014-9301-3.
- **Sharma and Jain (2017):** Sharma, A., & Jain, A. (2017). A hybrid fuzzy-based neural network approach for software defect prediction. *Procedia computer science, 115*, 319–326. https://doi.org/10.1016/j.procs.2017.09.109

- **He et al. (2013):** He, P., Li, B., Liu, X., & Ma, Y. (2013). Software defect prediction using tree-based ensembles. *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering*, 1–8. https://doi.org/10.1145/3416508.3417114.
- **He, Peng, et al**. "An empirical study on software defect prediction with a simplified metric set. "Information and Software Technology 59 (2015): 170-190.
- **Khoshgoftaar et al. (2014):**Khoshgoftaar, T. M., &Gao, K. (2014). Ensemble learning for software defect prediction. *Journal of Systems and Software, 86*(1), 1–10.https://doi.org/10.1016/j.jss.2012.10.030.
- S. K. Cowlessur, S. Pattnaik, and B. K. Pattanayak, "A Review of Machine Learning Techniques for Software Quality Prediction," *Advanced Computing and Intelligent Engineering*, 2020.
- S. Omri and C. Sinz, "Machine Learning Techniques for Software Quality Assurance: A Survey," 2021.
- Machine Learning Based Methods for Software Fault Prediction: A Survey," *Expert Systems with Applications*, vol. 172, 2021.
- T. Sharma, M. Kechagia, S. Georgiou, et al., "A Survey on Machine Learning Techniques for Source Code Analysis," 2021.
- A. Khan, R. R. Mekuria, and R. Isaev, "Applying Machine Learning Analysis for Software Quality Test," 2023.
- N. Fred and I. O. Temkin, "A Systematic Literature Review on the Use of Machine Learning in Software Engineering," 2024.