CSE422: Artificial Intelligence

'Software Quality Checker'

Project Report

Group: 13, Section: 03

Prepared by:

Azmari Sultana (22201949)

B M Rauf (22201782)

Submitted to:

Syed Zamil Hasan Shoumo

Adiba Tahsin

Table of Contents

1.Introduction:

The objective of this project is to predict the level of software quality (Low, Medium, High) using quantitative and categorical metrics of software. The quality of software is a very important element in minimizing bugs, maintaining, and advancing reliability. The inspiration behind this project is to create models that can automatically label modules as either High, Medium, or Low quality so that developers can focus more on testing and refactoring work.

2.Dataset Description

2.1 Dataset overview:

The dataset with which this paper will work is a software quality dataset with 1600 records and 9 attributes. Predict the quality label of software modules, which may be one of three values Low, Medium, or High. Since the outcome variable is discrete and categorical, this is obviously a classification problem and not a regression task.

The characteristics are both quantitative and categorical:
Quantitative features: Lines_of_Code, Cyclomatic_Complexity, Num_Functions, Code_Churn, Comment_Density, Num_Bugs, Code_Owner_Experience
Categorical feature: Has_Unit_Tests (Yes/No)
Target variable: Quality_Label (Low, Medium, High)

Most machine learning algorithms do not allow working with categorical variables directly, so the column Has_Unit_Tests (1 to Yes and 0 to No) and Quality_Level (high 0, low 1, medium 2) were converted to numeric values. This makes the categorical information efficiently applicable by the models. We used the Seaborn library to apply the correlation heatmap to investigate the relationship between features and the target variable.

Findings:

There was a strong positive correlation between Lines_of_Code and Cyclomatic_Complexity as it should be as bigger codebases tend to have more branching logic.

Num_Bugs was positively correlated with Quality_Label at a medium level meaning that an increase in the number of bugs would be associated with a decrease in perceived software quality.

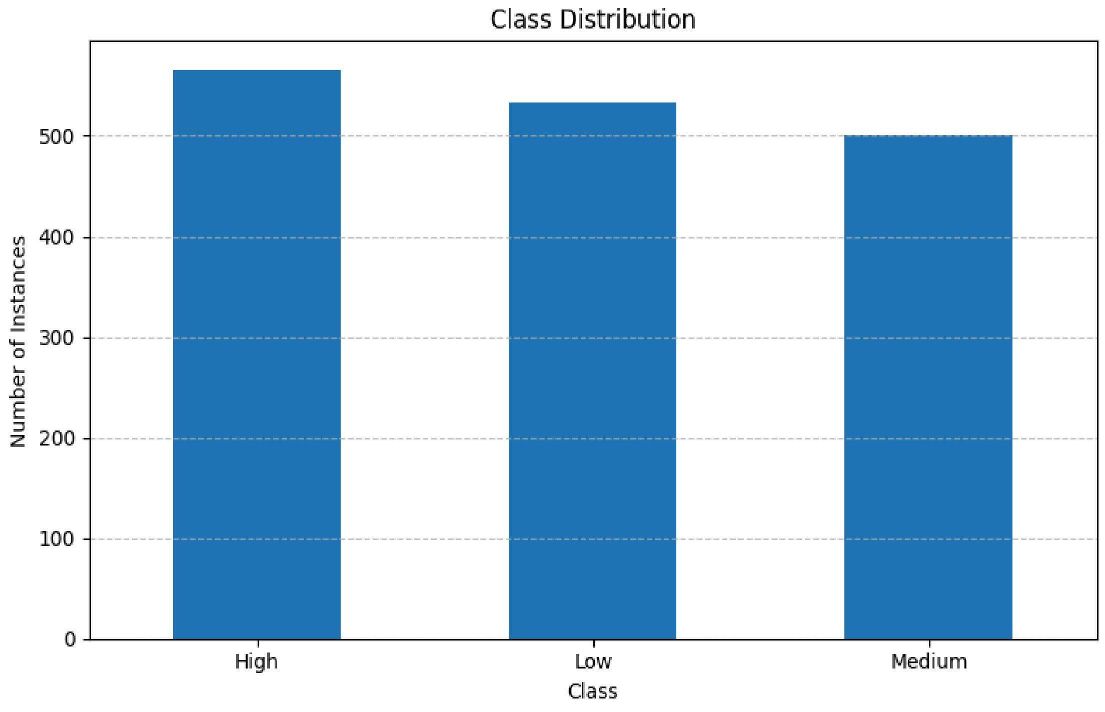
Has_Unit_Tests exhibited an oblique relationship with quality: test projects are more likely to have moved into High quality labels.

Other characteristics were less correlated and may be secondarily associated.

2.2 Imbalanced Dataset

The target variable Quality_Label was represented with the help of a bar chart to visualize the distribution. The findings revealed that the three classes (Low, Medium, High), are not equally represented and thus a problem of class imbalance.

Imbalance in the dataset may be biased to favor the majority class at the expense of the minority classes. Even if there is an imbalance, since they are very close, such a small imbalance will not be a problem.



2.3 Exploratory Data Analysis

Based on the exploratory analysis, some relationships with importance were drawn:

Very large and complex codebases are typically represented by the ‘Low’ quality category of modules based on their Cyclomatic_Complexity and Lines_of_Code.

Better quality labels are associated with higher Num_Bugs, which makes sense as the intuitive quality idea that bug-prone software has low quality.

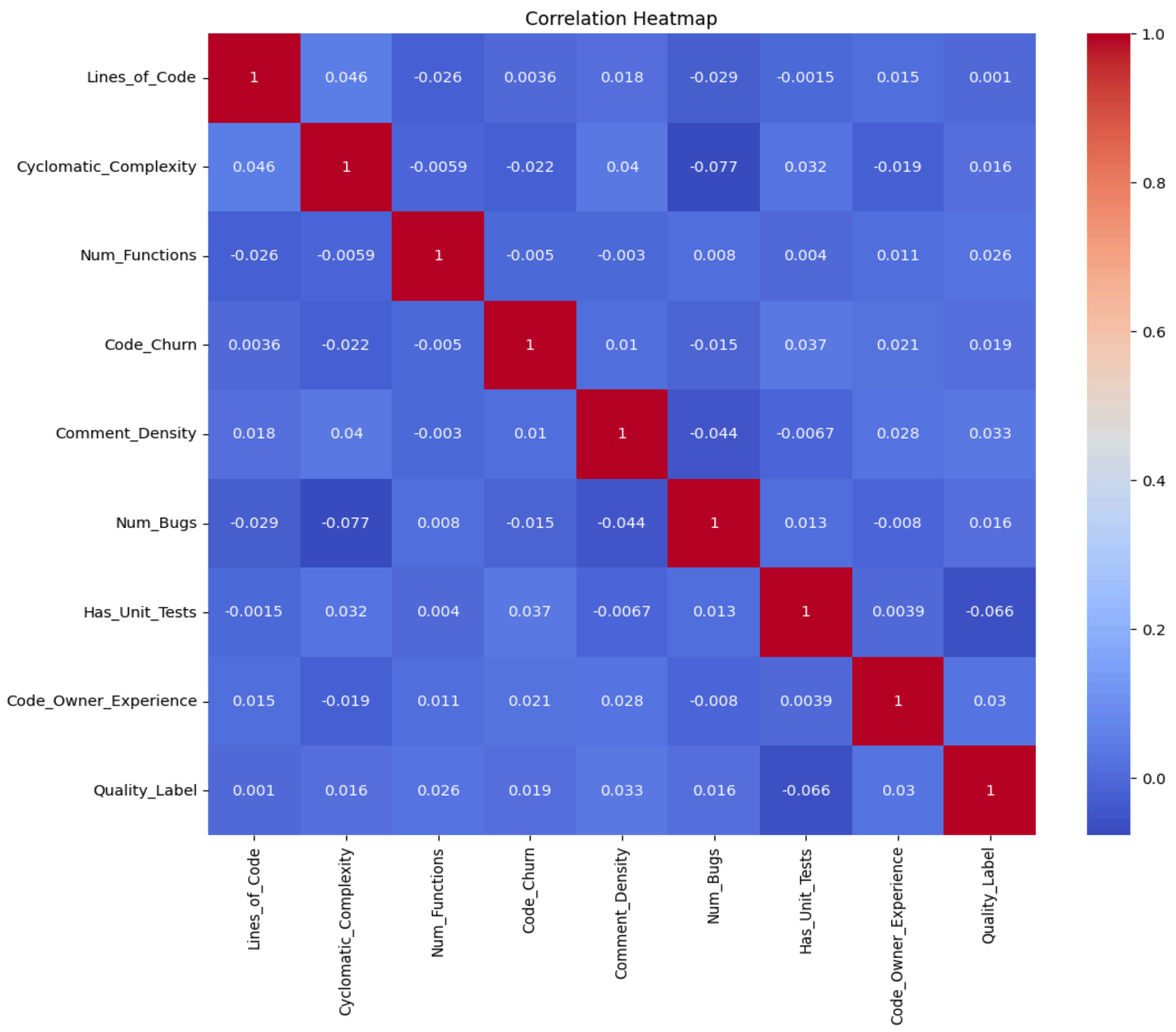
The Projects Has_Unit_Tests = Yes belong to the category of Medium or High quality, which also indicates the effects of testing practices on quality.

The Code_Owner_Experience seemed to have a positive impact on quality; more experienced owners created more quality modules.

In general, the data contains valuable predictive variables, yet it has issues with an uneven distribution of classes and predictor interrelations, so it is addressed in the pre-processing and the model training phase.

3.Dataset Pre-processing

There had to be preprocessing done so that the dataset is clean, consistent and can be used to apply machine learning algorithms. The raw data was found to have multiple issues and solutions to these issues were implemented.



Handling Missing Values

Problem:

Certain numeric values like Lines_of_Code, Cyclomatic_Complexity and Code_Churn had missing values. As the majority of machine learning models in scikit-learn processes do not directly accept missing entries, this had to be addressed.

Solution:

In the case of the numeric features, missing values were filled in with a median strategy. This minimizes the effect of extreme outliers contrasted with the use of mean imputation.

None of the features had more than half the entries, hence none of the columns were dropped.

Categorical Variables

Problem:

The Has_Unit_Tests was a categorical feature set as either Yes or No. String labels cannot be directly processed with most algorithms.

Solution:

Applied binary: Yes 1, No 0.

Quality level: High 0, Low 1, Medium 2

This holds the categorical difference but does not restrict models to numeric interpretation of the feature.

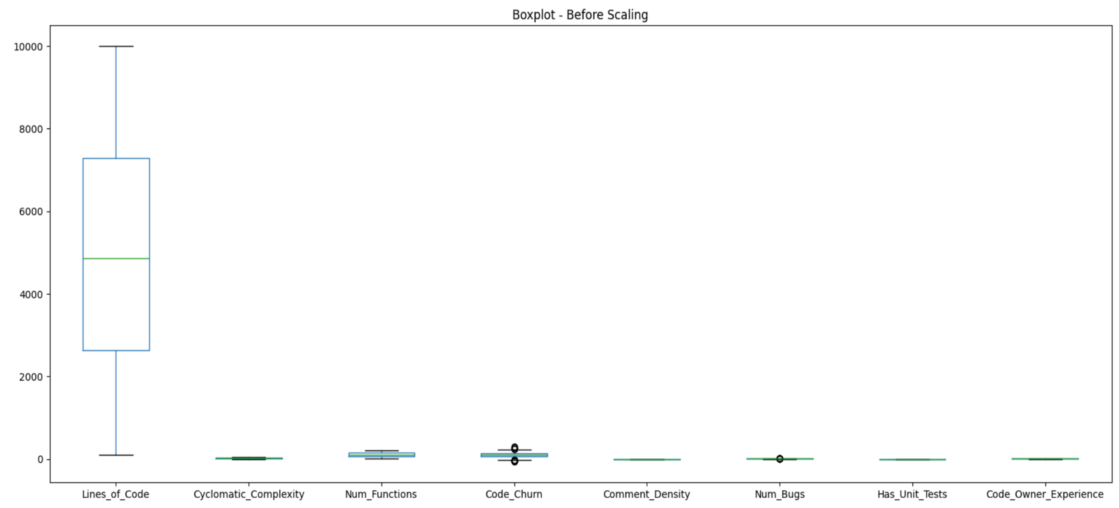
Feature Scaling

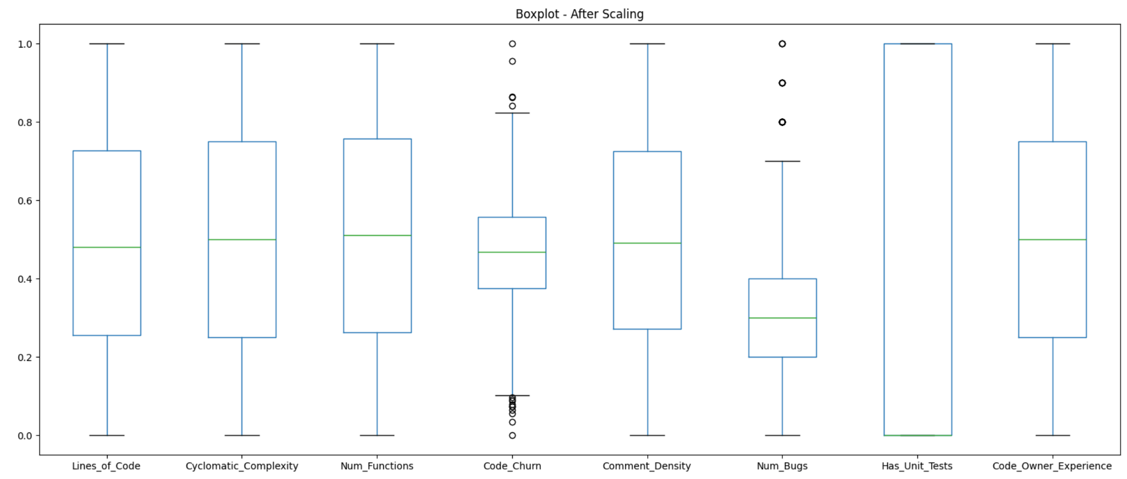
Problem:

There is large range variation and outliers in the dataset. For example, Lines_of_Code with values up to the thousands and others between 0 and 1. Then, outliers such as in the features of Code_Churn and Num_Bugs may tend to confuse the models due to inflated variance and decision boundaries

Solution:

These values were not deleted, but were dealt with indirectly in scaling. StandardScaler was applied to standardize all the numeric features on the same scale with mean equal to 0 and variance equal to 1. This made sure that other features did not prevail over the other in terms of their size.





4.Dataset Splitting

- Stratified Random train-test split (to preserve class balance).
- Training set: 70%
- Test set: 30%

5.Model training & testing

We compared three supervised learning algorithms to categorize software modules as low quality, medium quality, and high quality software. All models were trained and assessed on the 30 percent held-out test set and 70 percent of the data. The performance of each of the models was measured in terms of accuracy, macro-averaged precision, recall and F1-score to allow making a fair comparison, as they were all trained on an equal dataset.

5.1 Decision Tree

A supervised learning algorithm that is used in classification and regression problems is a decision tree. In decision tree decompositions data may be split into subsets according to the feature that yields the greatest information gain or least impurities (E.G., GINI index or entropy). Each internal node has the form of a choice depending on a feature, branches indicate an outcome, and leaf nodes indicate a final prediction.

The model was 32.08% accurate and macro-averaged precision, recall, and F1-scores were approximately equal to 0.32. The confusion matrix showed all three quality classes had misclassifications but the model could pick up some structure in the data. Generally, the Decision Tree was a good starting point, because it is easy to interpret and can work with numerical as well as categorical data.

5.2 Naive Bayes

Naive bayes is a probabilistic classifier using the theorem of Bayes, which makes the assumption of independence of features. In a number of real-life situations, particularly in text classifier and spam filtering, it surprisingly works well despite its simplicity and its naive assumption that features are independent.

It was the most accurate model (35.42%), with a macro-F1 of 0.34. It was found that although Naive Bayes performed well when predicting certain classes, it was not capable of separating Low and Medium quality categories because of overlapping.

5.3 Neural Network

MLPClassifier was used to implement a simple feed forward neural network.

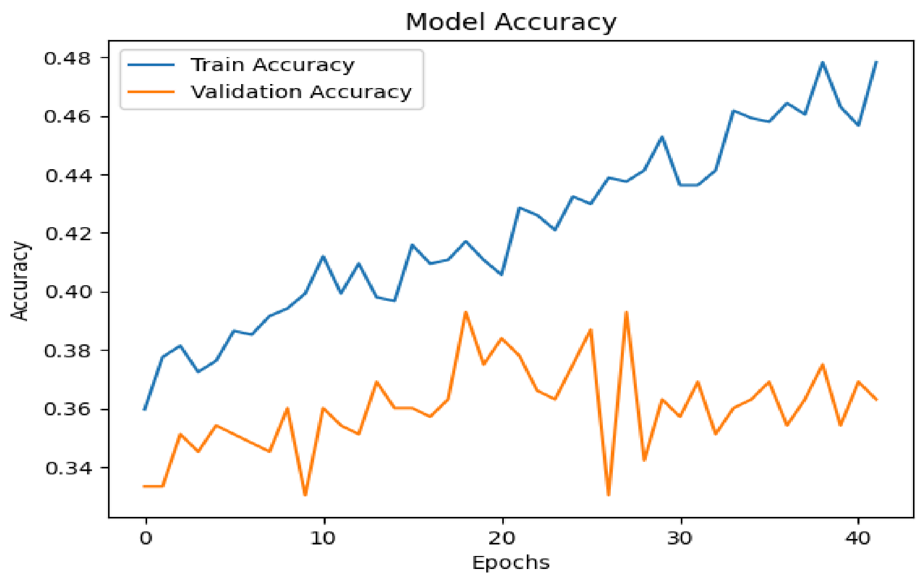
Architecture: There is a single hidden dense layer (containing 32 ReLU neurons) and an output layer (with three softmax units).

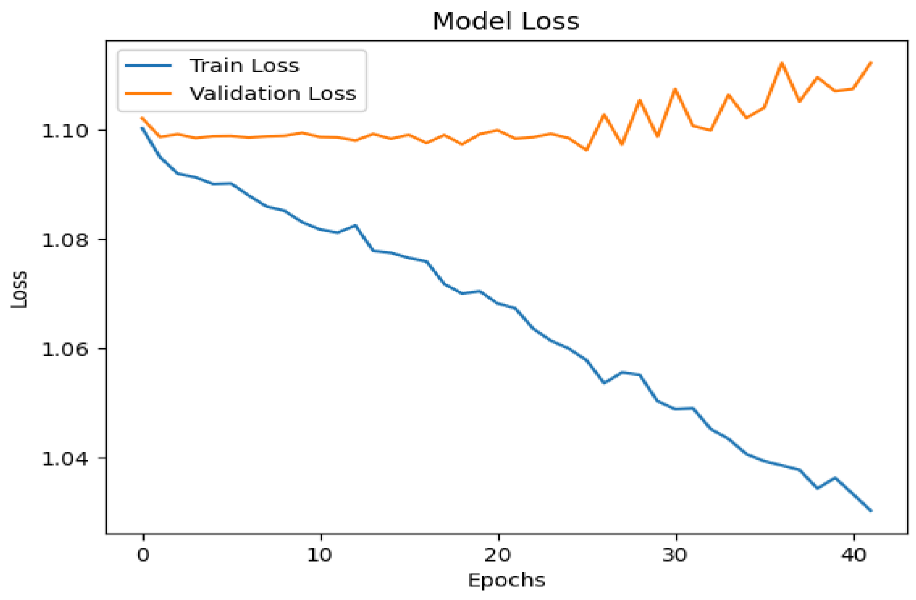
Training configuration: Adam optimizer, with a maximum of 500 iterations.

The neural network was the least accurate with an F1-score of 0.29 and had an accuracy of 29.79%.

It was likely the poor performance of the dataset: class imbalance and a relatively small sample size, which did not allow the model to generalize well.

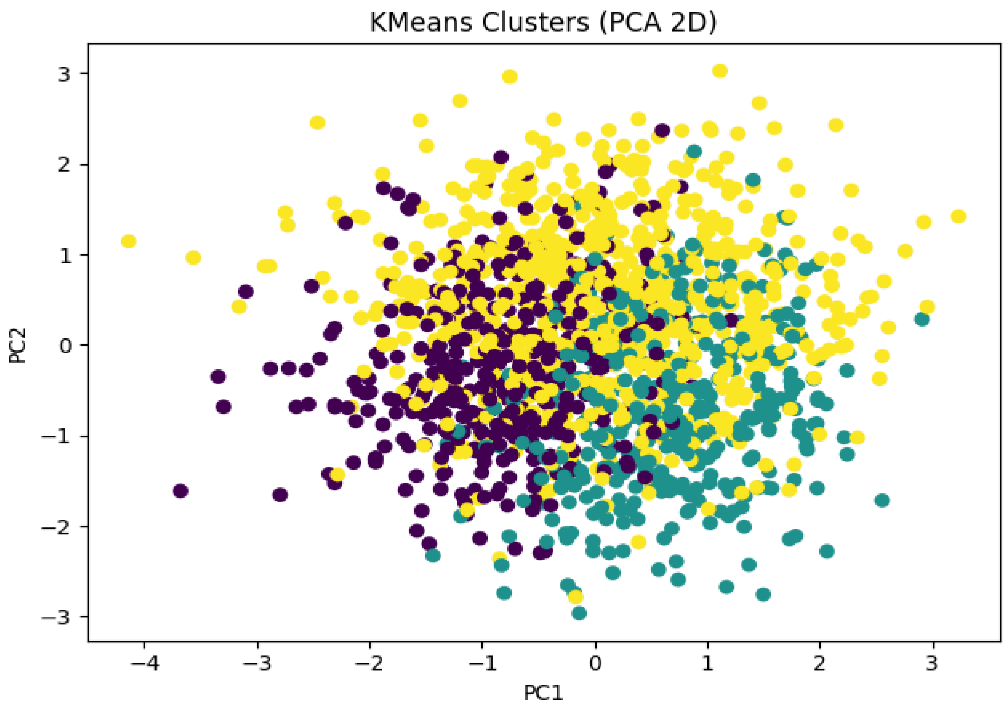
Neural networks can model complex non-linear relationships but to do so they need larger and more balanced datasets to realize their potential.





5.4 Unsupervised Clustering

We used K-Means clustering of the standardized data with k = 3 which corresponded with the quality categories (Low, Medium, High). The performance of the clustering was very poor and an average silhouette score of near 0.2 indicated that there was no great separation between the groups. When the clusters were compared against the actual Quality Label, it was found that one cluster had more Low quality modules, the other two clusters were more of a mix with Medium and High quality instances nearly evenly distributed. This means that the features distinguish some difference between Low and not Low quality but do not distinguish between Medium and High. The result is in line with the supervised outcome: even the most accurate model (Naive Bayes with 35.42% accuracy) did not divide well between the Medium and the High, and other models, Decision Tree (32.08%) and Neural Network (29.79%) also demonstrated the same weaknesses.



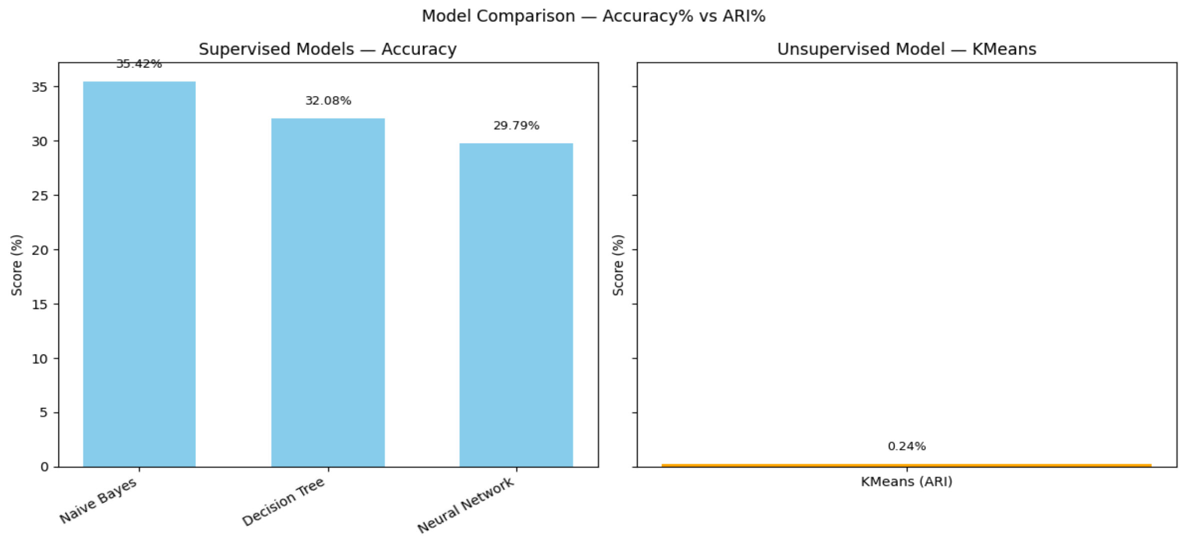
6.Model Selection

Decision Tree achieved the best results in general with Naive Bayes classifier coming second. The Neural Network was the one with the lowest accuracy and macro-F1 score.

Despite the relatively better performance of Decision Tree, all models proved to be unable to differentiate between the three quality levels, and many misclassifications were observed in the confusion matrices. Difficulties include the heavy overlap in feature distributions and small size of dataset.

Accuracy Comparison

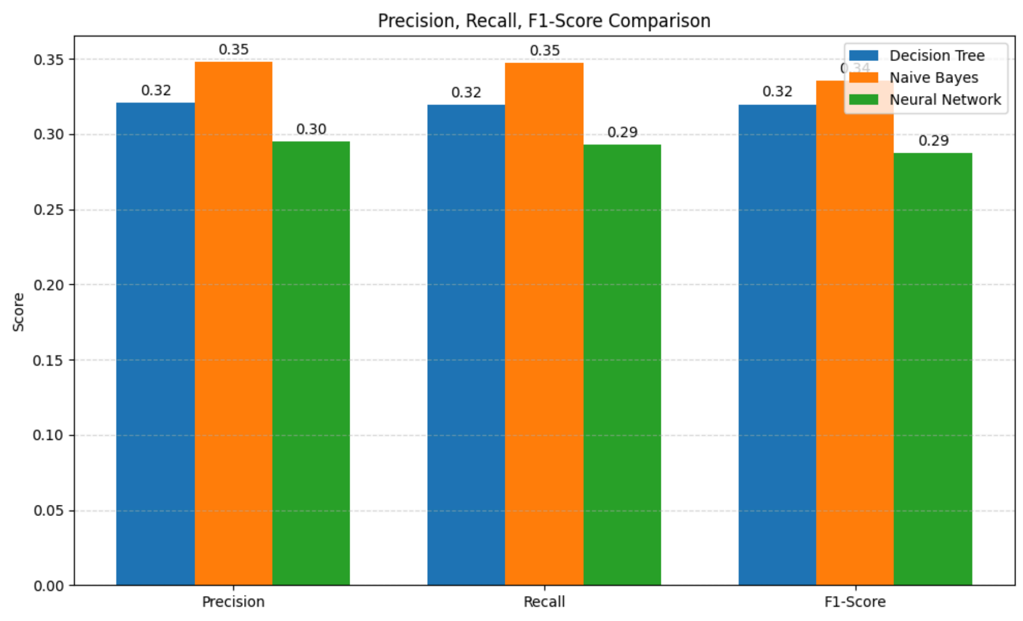
The following bar graph shows the accuracy of prediction between the models. The Decision Tree achieved the best test accuracy with Naive Bayes in second followed by the underperforming Neural Network.



F1, Recall and Precision Comparison.

Macro-averaged precision, recall and F1 are stated to explain the unbalancing between Low, Medium, and High quality classes.

Decision Tree continued to do better compared to both Naive Bayes and Neural Network although the improvements are relatively small.



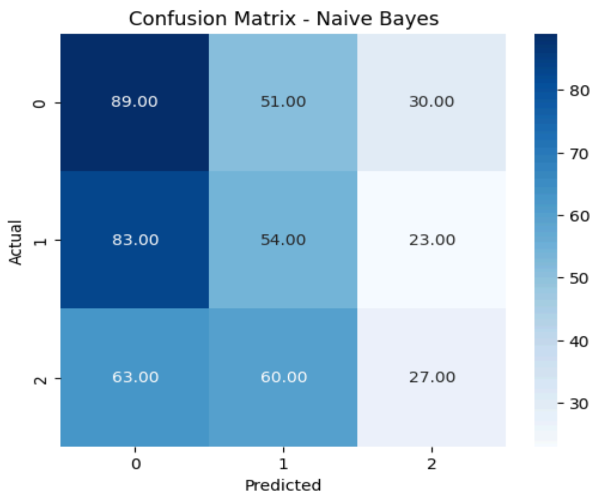
Confusion Matrices

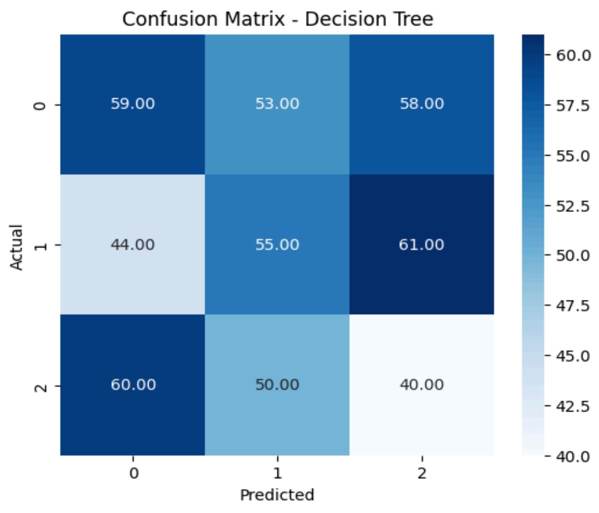
The confusion matrices also show certain trends in misclassification:

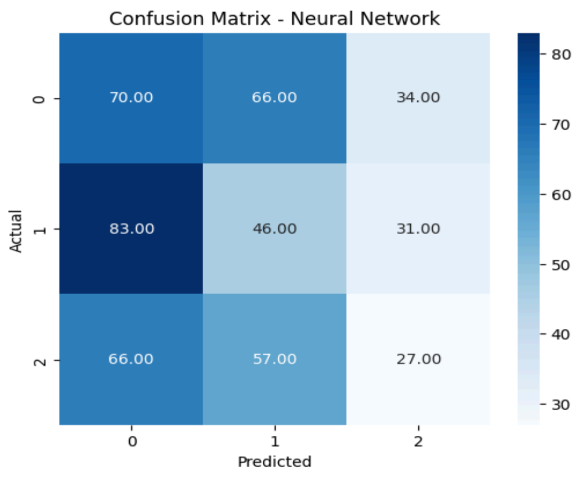
Decision tree: Will be more sensitive to Low quality modules than the other two, but mixed up most of the time between Medium and High.

Naive Bayes: There is a high misclassification in all classes because of the independence assumptions.

Neural Network: Failed to learn, there were scattered misclassifications in the categories.

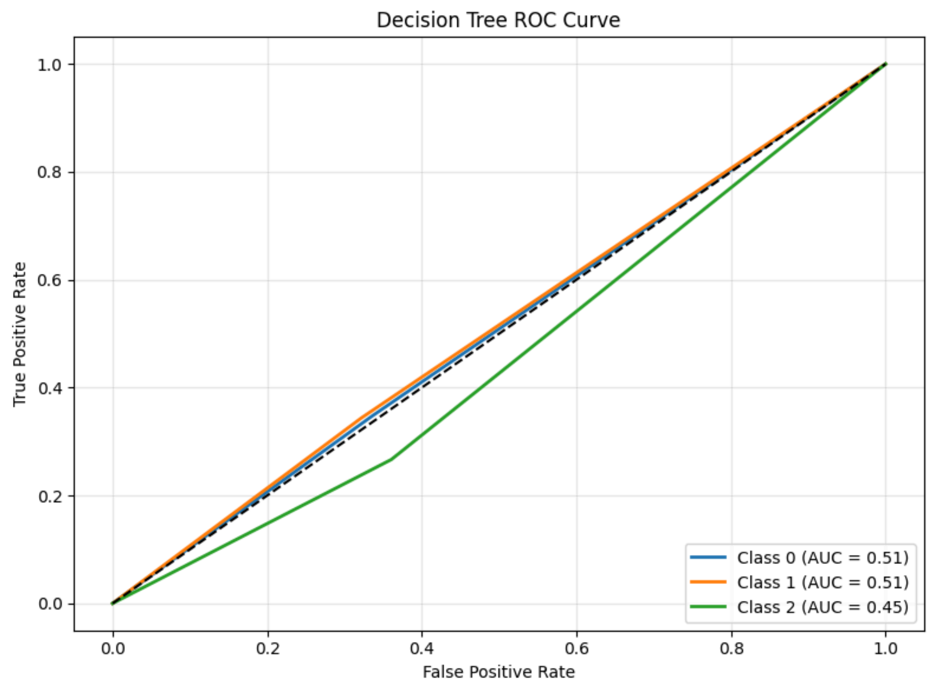


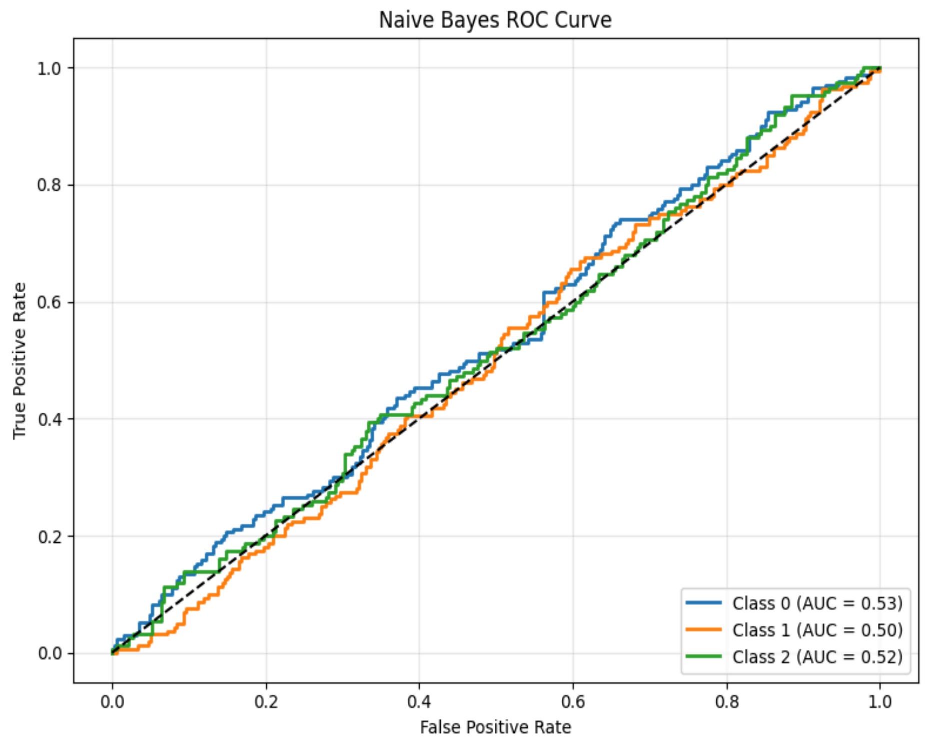


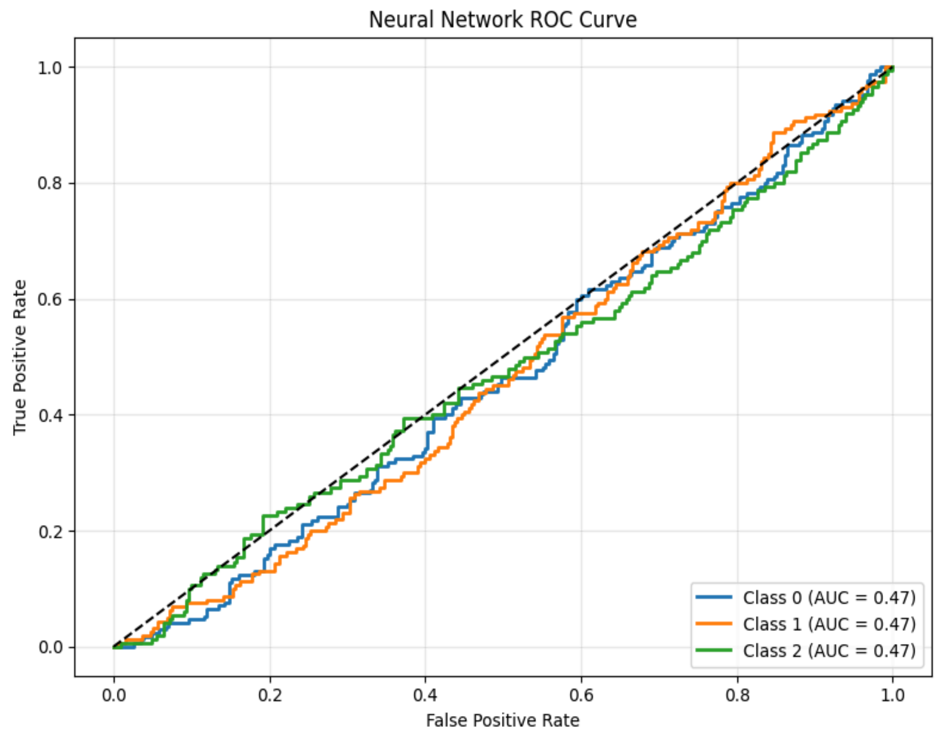


ROC Curves and AUC

Each model was used to produce one-vs-rest ROC curves. The Decision Tree had the highest area under the curve (AUC), Naive Bayes had a slightly lower area under the curve, and Neural Network had the lowest area under the curve, all confirming that the probability calibration was poor.


Decision Tree ROC Curve
True Positive Rate
False Positive Rate
Class 0 (AUC = 0.51)
Class 1 (AUC = 0.51)
Class 2 (AUC = 0.45)


Naive Bayes ROC Curve
True Positive Rate
False Positive Rate
Class 0 (AUC = 0.53)
Class 1 (AUC = 0.50)
Class 2 (AUC = 0.52)



7.Conclusion

This project investigated the application of machine learning models to categorize software modules as low, medium, or high quality, according to the measures of the static code. The outputs indicate that the Decision Tree classifier performed the best overall, with the highest accuracy and macro F1 score, while Naive Bayes model was performing moderate, and Neural Network had difficulty converging with the available dataset. These results indicate that the non linear relationships captured by the Decision Tree enabled it to model software quality better than the simplifying independence assumptions of Naive Bayes and the poor capacity of the selected neural architecture did. The fact that data is relatively small, features overlap, and that the classes are slightly unequal also led to frequent misclassifications especially between high quality and medium quality modules.

The difficulty in managing correlated features, balancing evaluation, and getting successful convergence of the neural model demonstrated the complexity of predicting software quality based on static metrics. On the whole, the Decision Tree was the most appropriate model in this research but the outcomes indicate that it is desirable to use bigger datasets, consider more features engineering, and develop more advanced learning algorithms that can enhance predictive performance in the future.