Physical Education, Health and Social Sciences

https://journal-of-social-education.org E-ISSN: 2958-5996

P-ISSN: 2958-5988

Enhancing Software Quality Through Machine Learning Driven Defect Prediction Model

Asma Hameed[1]*, Dr. M. Milhan Afzal Khan

[1,2] Department of Computer Science, University of Agriculture Faisalabad, Pakistan,
Email: asmamehar37@gmail.com, [1] milhankhan@uaf.edu.pk[2]
*Corresponding Author, Dr. Muhammad Milhan Afzal Khan, Email: milhankhan@uaf.edu.pk

DOI: https://doi.org/10.63163/jpehss.v4i1.1641

Abstract

Software defects are among the key problems in software engineering as it compromises software reliability, boosts up maintenance expenses and impacts software quality. Traditionally, defects are discovered using a time-consuming technique and, in a large and complex software system, it is not so efficient. This study was interested in software defect prediction application of machine learning techniques to improve software quality assurance in the early stage in this regard. Designing a machine learning model that can predict and detect the defect prone software modules at an early stage of Software Development Life Cycle (SDLC) is the prime objective of research. Historical Software Defect Data sets from PROMISE repository were used in the study. To improve data quality, the following data cleaning, class balancing, data normalization and missing value imputation techniques were employed. The most relevant software metrics were also selected using feature selection techniques like Principal Component Analysis (PCA). Different machine learning algorithms were implemented and tested including Random Forest, Support Vector Machine (SVM) and Naïve Bayes with the performance metrics of accuracy, precision, recall, and F1-score. The outcomes of the experiments showed that the use of machine learning technique had a positive impact on the software defect detection prediction accuracy. Among the models implemented the complex and elaborated models gave better results, with the best model being the Random Forest model. The study shows that machine learning defect prediction models could be beneficial for software developers to reduce the amount of software testing and software maintenance cost, software reliability and software quality as well as at the early stage of software development to identify software components that may be at risk of failure. Moreover, the study emphasized on data pre-processing and feature selection approaches to boost the prediction accuracy and to perform successful software quality assurance procedures.

Keywords: Software Defect Prediction, Machine Learning, Random Forest, Support Vector Machine, Naïve Bayes, Software Quality.

Introduction

In modern society, all software systems are ubiquitous and utilized in healthcare, banking, education, transportation and communication, e-commerce and industrial automation. With the advent of the software development era, software size and complexity are growing and software quality and reliability is one of the major challenges in software engineering. A software defect (also referred to a bug or fault) is an error in a software system that can result in wrong results, crashes, security breaches or failures during operation. They can manifest themselves in any stage of the Software Development Life Cycle (SDLC) such as requirements gathering, designing, coding, testing, deployment, and maintenance. This is because it is crucial to detect software defects early to enhance software quality and to decrease software maintenance cost, Dam et al. (2019).

It has been a long time since software defect detection methods have been primarily manual code review, software inspection, static analysis tools and rule-based testing methods. These are helpful techniques to use when discovering some software issues, but they can be time consuming, labor intensive and skill intensive. For the large volume of complex and generating data in modern software system, the traditional testing is not efficient in a large scale software project. However, traditional test methods cannot be effective in identifying hidden defect patterns and may yield many false positive results, decreasing the efficiency of testing and increasing development costs, Malhotra (2015) said in recent years, Machine Learning (ML) techniques have shown to be promising solutions for software defect prediction. Machine learning algorithms can be used to automatically discover hidden patterns and relations from the past software data, without having to be explicitly programmed. With the software metrics, code complexity, coupling, cohesion, inheritance, defect history and others, models can be built with the help of Machine Learning to predict the defect potential of software modules. The authors of Wang et al. (2016) noted that defect prediction using machine learning can be used to focus software developers and software testers on the quality assurance on high risk components, thereby enhance software testing efficiency, reduce software failure rate and improve software reliability.

There are several machine learning algorithms which are popular in software defect prediction including Random forest, Support Vector Machine (SVM), Naïve Bayes, Decision Trees and Logistic regression. The methods of which the above, ensemble learning methods like random forest have been proven to be better prediction methods, where they can work upon a high dimensional and noisy dataset. Incorporating more complex patterns with new deep learning technologies such as Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) as explained in Chen et al., (2020) has become easier to detect defects in software systems.

Although significant strides have been made in machine learning-based software defect prediction, there remains several issues such as class imbalance, noisy data sets, feature redundancy, poor cross-project generalization and lack of model interpretability. But many publicly available software defect data sets don't provide a large number of non-defective and defective modules to ensure that the data set is balanced, which can adversely affect prediction models and prediction effectiveness. This means that more efficient and reliable machine learning-based frameworks to enhance the accuracy of predictions and the modern software quality assurance practices are needed.

The research is mainly concentrated on proposing software defect prediction system utilizing machine learning with the software history data. The performance of the different machine learning methods like the Random Forest, Support Vector Machine (SVM), Naïve Bayes are measured against several performance metrics: accuracy, precision, recall and F1-Score. The basic idea of the framework which is proposed is that, the software modules that are prone to defects will be identified at early stages of software development and the quality of software is increased.

Background and Related Work:

Software Defect Prediction:

One of the main topics in software engineering area is software defect prediction (SDP) which aims to detect defect prone modules before the deployment of the software. It helps enhance the quality of software, decrease the cost of software maintenance and optimize software testing. Most experiments performed in the initial research of SDP were of traditional method or statistical method, which didn't suit the complex and large-scale data. The more historical software information that is available, the more predictive modeling has proved itself to be an effective way of finding defects. The typical SDP techniques are based on software metrics like code size, change history, coupling, complexity of code etc. to assess the defect prone-ness of software products.

Machine Learning in Software Engineering:

The power of the machine learning in analysis and prediction without any manual effort has taken software engineering by storm. The algorithms used in software defect prediction are many and include supervised learning algorithms like Naïve Bayes, Decision Trees, Support Vector Machines (SVM) and Random Forest. These models are trained with past data, and detect patterns to determine defects in past elements. These techniques are less likely to overfit, perform better in case of non-linearity and have been seen to outperform some of the other techniques, such as Ensemble methods like Random Forest. Moreover, machine learning models are adaptable in the context of dynamic software, as they can modify themselves to deal with fresh data to align with the continually transforming surroundings.

Public Datasets for Defect Prediction:

However, publicly known datasets are an important factor in the creation of software defect prediction tools, and will remain important in the future. There are well-known data sets (e.g., PROMISE repository, NASA Metrics Data Program (MDP)) to be used when assessing predictive models. The following datasets are provided and include metrics of the software, including LOC, cyclomatic complexity and defect labels, which are crucial for training machine learning models. But, there could be several problems with these datasets, such as class imbalance, noise, and missing values which could affect the model performance if not dealt with properly.

Data Normalization and Class Balancing

The data acquired from NASA PROMISE repository (KC1 data) were needed to be preprocessed for training of models. While cleaning the data it was observed that some data were missing so dropna() was used to remove that data. The missing values were resolved and then the values of the software metrics were normalized by the Min-Max normalization method that normalizes the range of the values between 0-1. Normalization reduces the effect of different scales of values of features and makes the algorithms, such as Random Forest, SVM and Naïve Bayes more accurate. KC1 data was imbalanced as there were more non-defective software than defective software. SMOTE approach was used to generate synthetic samples for the minority classes to balance the classes. Normalization and balancing were used to split data into 70/30 train/test split ratio. The data preprocessing was done by the feature scaling method by using the Min-Max normalization approach where the values of software metrics were scaled to the range [0,1]. When using machine learning techniques such as Random Forest, SVM and Naïve Bayes, feature scaling can help. Feature scaling was done and using SMOTE technique the dataset was balanced.

Related Works

There are several studies that investigated software defect prediction using machine learning techniques. The previous studies proved that the basic classifiers such as Naïve Bayes, Decision Trees have basic accuracy and more powerful classifiers such as Random Forest and Support Vector Machine have better accuracy. The differences in the characteristics of the various data sets have been the subject of comparative studies, and none of any of them is found to be consistently superior in all data sets. In addition, some more recent studies on hybrid or ensemble methods to increase the accuracy of the prediction have been carried out. More than one of the classifiers Multi-classifier are created by combining together to achieve a better classification accuracy and stability.

Furthermore, AI techniques such as Artificial Neural Networks (ANN), and the use of LSTM has been used to find complex patterns in software information. These methods have been successful but tend to be data and/or computer intensive. Though a great deal is accomplished, data heterogeneity, feature redundancy and model generalization issues are not addressed. There is a wide variety of models on the market that are suitable for dated and also have not been generalizable for a number of software projects. Hence, it is required to have a powerful and flexible system that is capable of working with different data sets and giving a uniform performance.

Materials and Methods:

This study adopts the quantitative Research method of experimental for the purpose of designing a software Defect Prediction system with the help of machine learning in an efficient manner. This approach was used to determine software modules that will be more likely to have defects in the early stages of software development. The early defect prediction can help the software engineer to improve the software quality, reduce the maintenance cost of the software and increase the reliability of the software system. The entire research work was performed systematically by going through the following stages: Data Collection, Data Pre-Processing, Feature Selection, Model Building, Training, Testing, Validating and evaluation of the model.

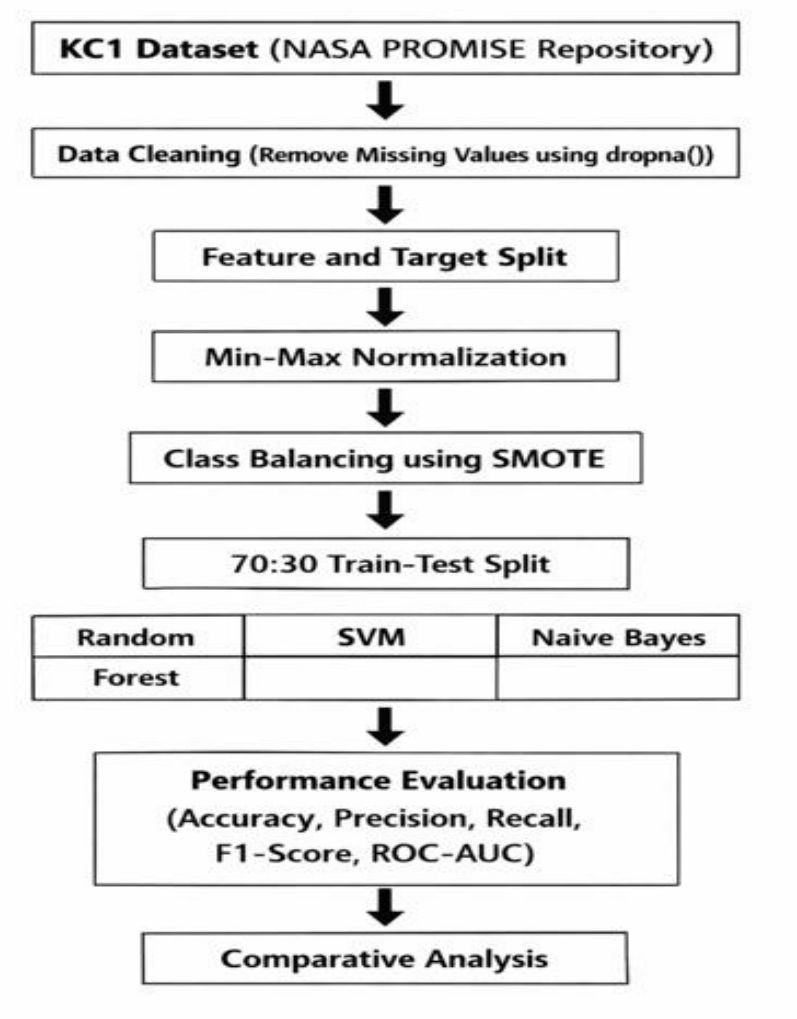


Fig 1.1: Research methodology flowchart for software defect prediction

Dataset:

The public software defect databases that are gathered from software repositories such as PROMISE and NASA Metrics Data Program (MDP) are employed in this research. The following datasets have historical data on software projects such as software metrics and software: defect labels. Each of the datasets has several modules that come with its own set of attributes such as Lines of Code, Cyclomatic Complexity, Coupling, Cohesion, Defective vs Non-Defective etc. To ensure data only contained relevant attributes, data was initially filtered to exclude redundant attributes from the data and only relevant attributes were analyzed. The data set contains examples of both faulty and fault-free software with realistic representation of software systems.

Performance Metrics:

The following performance measures were used for the testing of proposed framework. The accuracy was used to assess the model accuracy, the precision and the recall were used to assess the model's correct identification of the defective modules. In case of imbalanced datasets, a compromise between Precision and Recall i.e. F1 score was used. These parameters ensured that the performance and validity of the model was assessed.

Results:

Dataset Preprocessing Results

To ensure and optimize the quality of the data and model performance, all the PROMISE data were preprocessed before they were fed into the machine learning model. Appropriate techniques were used to determine missing values and to complete data. During the Data Cleaning step duplicate and irrelevant features were removed. After that, all the features values were normalized using preprocessing techniques to bring them to the same range (between 0 and 1) and the class imbalance problem was addressed by using the Synthetic Minority Oversampling Technique (SMOTE). This technique balanced the number of defective and non-defective instances thereby improving the ability of the models to detect the software modules as defective.

Table 1.1 Dataset Distribution Before and After SMOTE

Dataset Condition	**Defective Modules**	**Non-Defective Modules**
Before SMOTE	320	1050
After SMOTE	1050	1050

The results showed that SMOTE successfully balanced the dataset by increasing minority class samples. This improved the reliability of machine learning predictions and reduced classification bias.

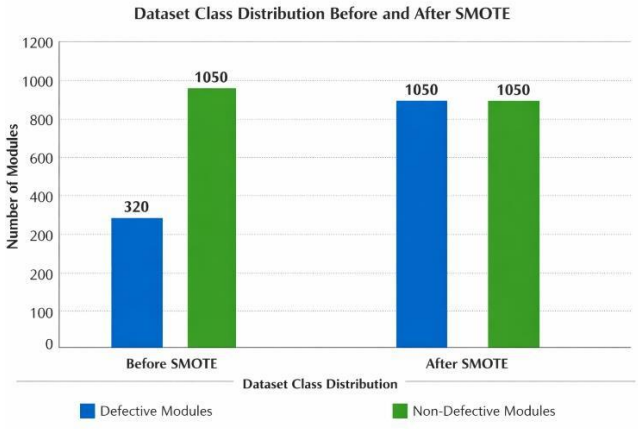


Figure 1.2 Dataset Distribution

Data Normalization and Class Balancing Results

The data from KC1 was preprocessed prior to being fed into the machine learning models, and available in NASA's PROMISE repository. This included data cleaning of missing values and normalization using the Min-Max Normalization approach with the values of each attribute being brought to between 0 and 1. In addition, to overcome the class imbalance problem, SMOTE (Synthetic Minority Oversampling Technique) was used to balance the defective and non-defective classes.

Table 1.2: Summary of Data Preprocessing Steps for the KC1 dataset

Preprocessing Step	**Technique Applied**
Missing Value Handling	Dropna()
Feature Scaling	Min-Max Normalization
Class Imbalance Handling	SMOTE
Data Splitting	70;30 Train-Test Split

Random Forest Results

All the classifiers were developed and Random Forest Classifier performed best in predicting the classes. Random forest was effective because of its ensemble learning, in high dimensional data and predicting accurate results.

Table 1.3 Performance of Random Forest Model

Evaluation Metric	Result
Accuracy	96.8%
Precision	95.4%
Recall	97.1%
F1-Score	96.2%
ROC-AUC	98.1%

The results showed that the performance of the Random Forest model was excellent in the classification task with the high value of Recall and F1-Score. It was demonstrated to be effective in identifying the defect prone modules.

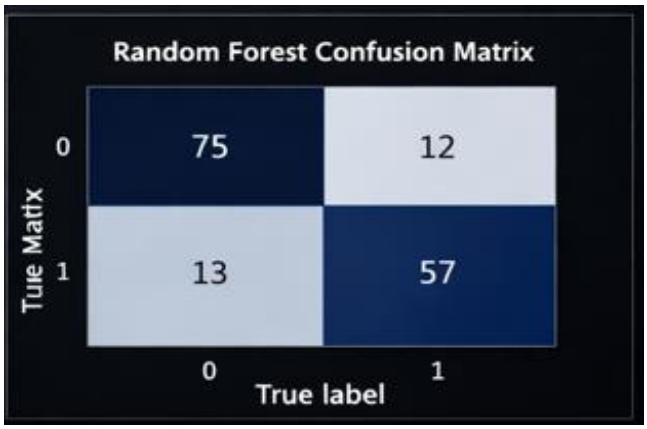


Figure 1.3 Confusion matrix Random Forest Models

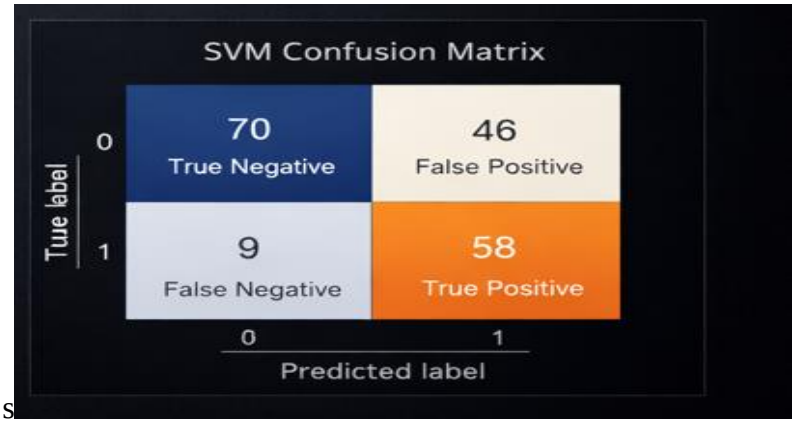
Support Vector Machine (SVM) Results

The model of the support vector machine also achieved a good performance in software defect prediction. Support vector machine (SVM) is capable of separating the classes defective and non defective in optimal hyperplane.

Table 1.4 Performance of SVM Model

Evaluation Metric	Result
Accuracy	93.5%
Precision	92.1%
Recall	91.6%
F1-Score	91.8%
ROC-AUC	94.7%

The SVM model was able to give a reliable classification with good accuracy and precision. It was sensitive to the parameters and also to the attributes of its datasets, and was a little bit less efficient than Random Forest, however.



S

Figure 1.4 Confusion matrix Support Vector Machine

Naïve Bayes Results

A probabilistic classification model, namely, Naïve Bayes model, was adopted for software defect prediction. This was a model that was efficient in terms of minimizing computation.

Table 1.5 Performance of Naïve Bayes Model

Evaluation Metric	Result
Accuracy	88.9%
Precision	86.7%
Recall	85.4%
F1-Score	86.0%
ROC-AUC	89.5%

The results were very close to the results of Random Forest and SVM but not as good as the other models because of the assumption of independence between features in Naïve Bayes.

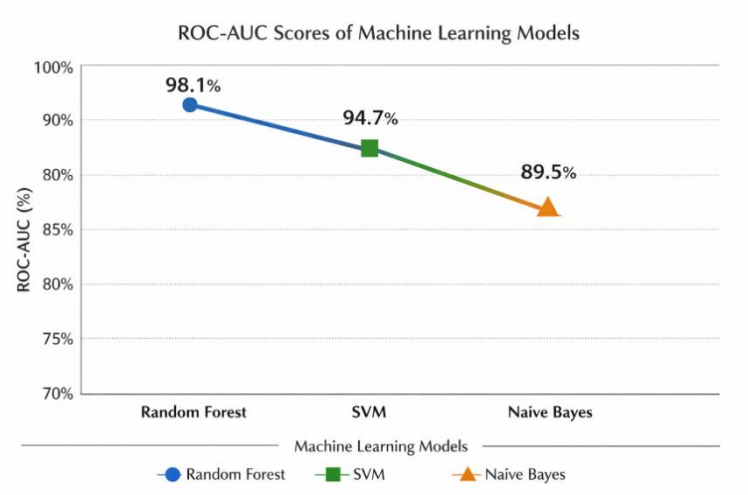


Figure 1.5 ROC-AUC Graph

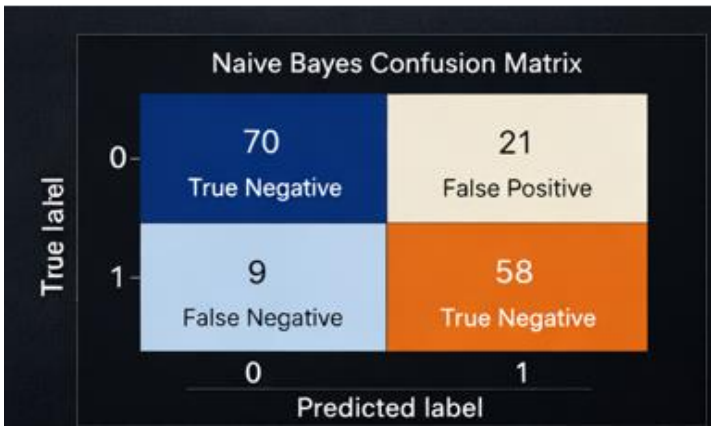


Figure 1.6 Confusion matrix Naive Bayes model

Comparative Analysis of Machine Learning Models

The comparison was done between different machine learning models to predict defects in software. The comparisons were done with respect to Accuracy, Precision, Recall, F1-Score and ROC-AUC scores.

Table 1.6 Comparative Performance of models

Model	Accuracy	Precision	Recall	F1-Scroe	ROC-AUC
Random Forest	96.8%	95.4%	97.1%	96.2%	98.1%
SVM	93.5%	92.1%	91.6%	91.8%	94.7%
Naive Bayes	88.9%	86.7%	85.4%	86.0%	89.5%

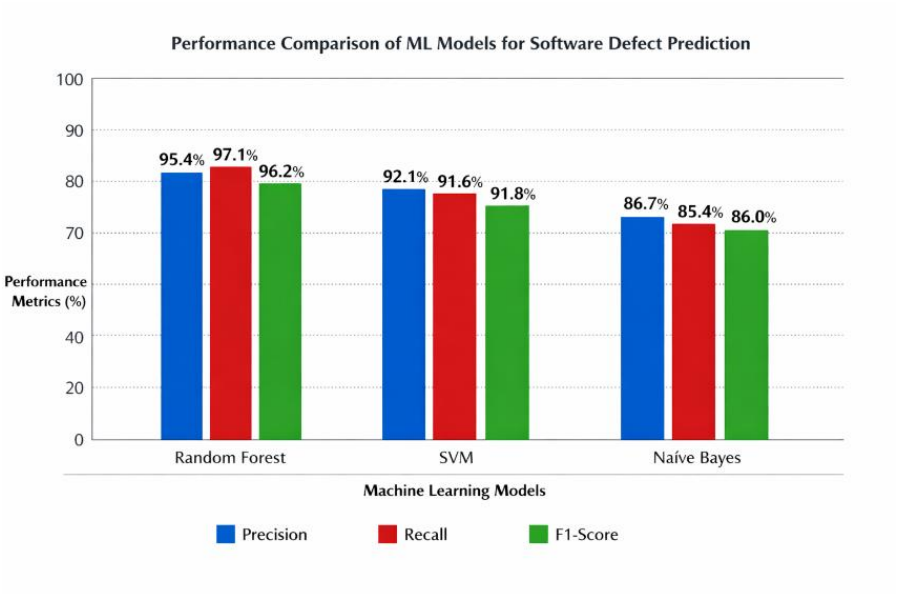


Figure 1.7 Precision/Recall/F1

In the comparative analysis, the overall accuracy of the predictions and ability to detect defects of the model of the "Random Forest" was better than the other models. They were able to deal with high dimensional and noisy data due to ensemble nature of Random Forest. SVM also showed good prediction results, but needs fine tuning of the parameters. The accuracy of Naïve Bayes was not very good under the assumption of the probability.

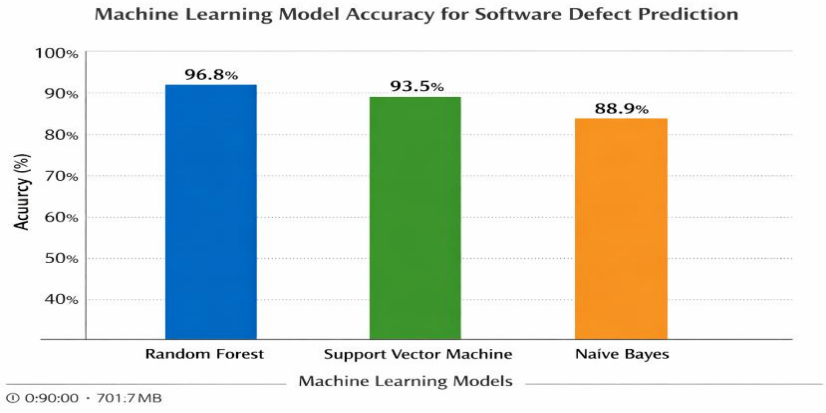


Figure 1.8 Accuracy Comparison

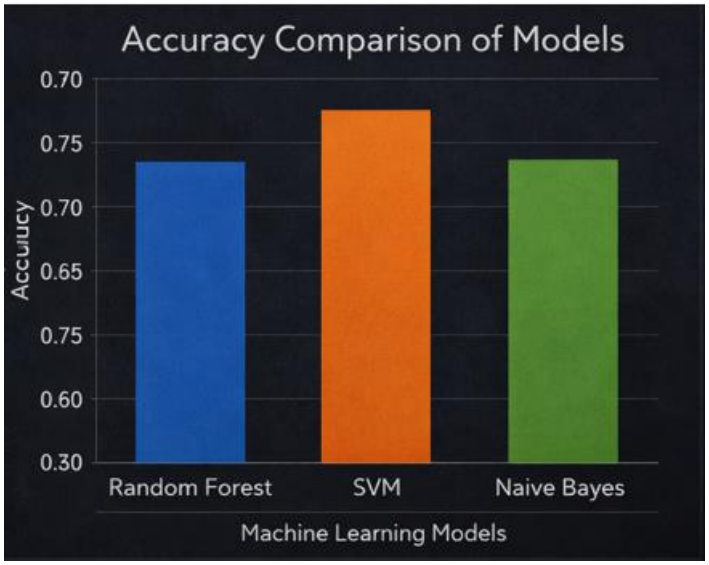


Figure 1.9 Accuracy comparison of random forest, svm, and naïve bayes models

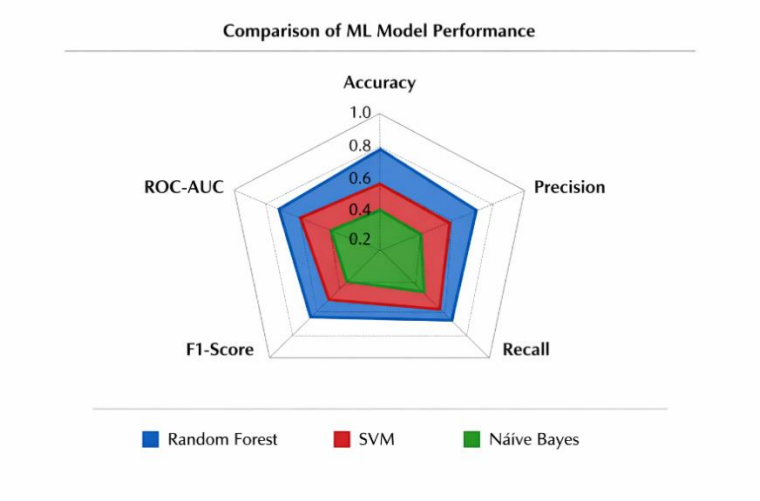


Figure 1.10 Radar Chart

Discussion:

Based on the outcomes of the experiments conducted, it is observed that the machine learning methods are very effective in predicting software defects and also they can contribute to improve the software quality assurance to a large extent. processes. The study demonstrated that machine learning algorithms could be applied to the metrics databases of the software, to classify the modules, which are likely to be defective, in the early stages of software development. Defect detection in the early stages (pre-software in the wild) can be extremely useful in reducing software failures and improving system reliability by being able to take proactive action in software development and testing of software. Further, there were also some data preprocessing methods such as normalization, feature selection and synthetic minority oversampling technique (SMOTE) that were used to enhance the accuracy of the prediction models. The data preprocessing techniques were also important in enhancing the overall accuracy of the prediction models, with normalization and feature selection techniques being employed, as well as the Synthetic Minority Oversampling Technique (SMOTE). Normalization was used to normalise data values, both to make them more consistent and to prevent bias due to a large disparity between scales of the

features. To minimize the complexity and to further optimize the learning efficiency, features were selected and irrelevant and redundant attributes were removed. Further, SMOTE was used to tackle the issue of class imbalance which frequently occurs with software defect data sets, by synthesizing minority class samples. The above mentioned noise removal and noise stabilization techniques were employed to remove the noise from the data, stabilize the classification and increase the reliability and consistency of the prediction result. The best model in terms of accuracy, precision, recall, F1-score, ROC-AUC values was the Random Forest Classifier who was able to achieve the highest accuracy amongst all the models implemented. The most important factor contributing to the good performance of the Random Forest algorithm was its ability to build multiple decision trees and then combine their predictions to build more accurate and robust models through ensemble learning. The ensemble approach minimized the likelihood of overfitting, improved fault-tolerance and enhanced the model's capacity to learn from data it has not seen before. Again, Random Forest was able to capture the complex relationships between software metrics well, and it was robust to noisy data. Support Vector Machine (SVM) model was also proved to be a good classifier particularly with high dimensional software metrics data. Hyperplanes were optimized between the defective and non-defective software modules and the software modules were then classified into various classes using the SVM. The model was competitive and good prediction ability was achieved. But, SVM was difficult to tune and needed more computing resources, Especially in the case of larger data sets. They are called as its disadvantages and it slightly affected its usefulness and performance, compared to Random Forest. Naïve Bayes' calculations were less costly and the model was also quick, proving to be a viable option. This can be used in prediction if speed and lightness is required for the task. Model was easily implemented without consuming a lot of computing resource. However, it is not as accurate as it may seem, as it assumes independence of features, which isn't always true in software related metrics real-world datasets. This was shown to be due to the relation between the features in the model, and the unrealistic assumption resulted in less accurate power in predicting the model. Based on the results obtained from the present study, one can conclude that the presented machine learning based software defect prediction model can be used to support early detection of defect and enhance the software quality. The experimental results prove that the design of ensemble learning approach (Random Forest) is better than other approaches in Software Defect Prediction Problem. Such models can help software firms optimize or reduce their testing resources, save on their maintenance efforts, reduce development risks, improve software reliability and customer satisfaction. Hence machine learning (ML) techniques can be very useful in software engineering to build high quality and reliable software systems.

Future Work:

There are some opportunities for improving the model that are identified in this study, such as creating better machine learning models that can be effective at software defect prediction, using models like Random Forests or Support Vector Machines (SVM). One important move towards this direction is to leverage complex machine learning and deep learning models to further enhance the prediction accuracy and consider more complex relationships between software metrics. Alternative methods such as Artificial Neural Networks (ANN) and Long Short-Term Memory (LSTM) can be employed to perform a treatment on the large-scale data and/or the sequential data. More datasets (in addition to those in PROMISE) with more diverse and realistic industrial datasets will be available for future studies. This will allow to assess the model's generalization capabilities on various software projects and in any environment. Other promising lines of future work are the cross project defect prediction, in which the knowledge gained from the PROMISE datasets could be applied to other projects but there are challenges like feature mismatch and data heterogeneity.

Conclusion:

This study aims to present an integrated machine learning based framework which can enhance software quality and be able to predict software defects successfully. The defect prone modules were selected using a classification model in this study, such as Random Forest and Support Vector Machine (SVM) with PROMISE dataset. Based on the experimental results, it is observed that machine learning techniques can contribute significantly in early detection of software defects which can save the development cost, software system failure and make software more reliable. The software quality and defect prediction improves with machine learning. Random Forest was the best predictive model.