Comprehensive Study on Machine Learning Techniques for Software Bug Prediction

Nasraldeen Alnor Adam Khleel, Károly Nehéz
Department of Information Engineering
Institute of Information Science
University of Miskolc
H-3515 Miskolc
Hungary

***Abstract*—Software bugs are defects or faults in computer programs or systems that cause incorrect or unexpected operations. These negatively affect software quality, reliability, and maintenance cost; therefore many researchers have already built and developed several models for software bug prediction. Till now, a few works have been done which used machine learning techniques for software bug prediction. The aim of this paper is to present comprehensive study on machine learning techniques that were successfully used to predict software bug. Paper also presents a software bug prediction model based on supervised machine learning algorithms are Decision Tree (DT), Naïve Bayes (NB), Random Forest (RF) and Logistic Regression (LR) on four datasets. We compared the results of our proposed models with those of the other studies. The results of this study demonstrated that our proposed models performed better than other models that used the same data sets. The evaluation process and the results of the study show that machine learning algorithms can be used effectively for prediction of bugs.**

Keywords*—*Static code analysis; software bug prediction; software metrics; machine learning techniques

I. INTRODUCTION

Due to the increasing size, complexity of software products and inadequate software testing no system or software can claim to be bugs free. There are many activities related to software testing such as implementing processes, procedures, and standards that must be carried out in a specific sequence to ensure that quality objectives are achieved or testing a product for issues such as software bugs. There are different classifications of bugs in software testing like Major defect: a defect, which will cause an observable product failure or deviation from functional requirements. Minor defect: a defect that will not cause a failure in execution of the product. Fatal defect: a defect that will cause application/system crash or close abruptly. Bugs can also be classified into functional defects, performance defects, usability defects, compatibility defects, security defects, etc. The use of analytical methods to check and review source codes is standard development practice. This process can be accomplished manually or automatically using static code analysis tools, dynamic code analysis tools, etc. Recently a lot of tools evolved for static code analysis, to provide a truly practical, value added solution to many of the problems that software development organizations face. But there are numerous false positives and false negatives results, which make these tools hard to be used in practice. So, there must be found another methodology or approach for static code analysis such as Machine Learning (ML) algorithms [1], [9], [12]. Software bugs usually appear during software development process. Software bugs are often difficult to detect or identify, and developers spend a large amount of time locating and fixing them. As well, some bugs cannot be detected at an early phase of development. To relieve the issue of bug fixing, the researchers did many extensively studies for bug prediction. Many machine learning (ML) driven prediction models have been built and tested on various basis. The process of software bug report is an important part of software maintenance, but the process of bug reports assignment can be very expensive in large software development projects, where a lot of studies suggest automating bug assignment approaches using machine learning in open-source software. Software Bug Prediction (SBP) plays a vital and important role in the process of improving software product quality. SBP is a process of generating machine learning models (classifiers) to predict software (code) defects based on historical data. The most recent methodologies used to predict software bugs are supervised(classification)machine learning models, and with recent advances in machine learning techniques, new models have emerged that have enhanced performance and capabilities in predicting software bug [2]. Classification is a major task of data analysis using machine learning algorithms that allow the machine to learn associations between instances and decision labels, from which an algorithm builds a model to predict the labels of new instances for a specific sample data. In machine learning, classification can be categorized into three types: binary (yes or no), multi-class, and multi-label classification [5], [25]. To build a dataset containing useful buggy code element characterization information, we chose Promise Repository dataset that stores software metrics along with bug information for many projects, these datasets were collected from real software projects by NASA [26]. The objective of this study is to investigate the previous studies that used most effective machine learning techniques for software bug prediction. In this paper, four supervised machine learning models are identified and utilized on four different datasets to evaluate the Machine learning algorithms capabilities in software bug prediction. The paper compares the proposed models based on various performance measures like accuracy, precision, recall, F1-score and ROC curves. The

structure of this study is organized as follow. Section 2 presents a discussion on software bug prediction by analyzing static code analysis. An overview of the machine learning techniques is presented in Section 3. After that, the literature review is presented in Section 4. Section 5 presents our research methodology. Section 6 presents software metrics and data sets. An overview of the selected machine learning classifiers and their evaluation is presented in Sections 7 and 8. Section 9 presents the experimental results and discussion followed by conclusions and future work in the Section 10.

II. Software Bug Prediction by Analyzing Static Code

Static code analysis is a method of analyzing software code without its execution to find potential problems like defects or bugs issues that might arise at runtime to check the quality of source code and addressing weaknesses in the program code through evaluating and correct source code based on some factors like structure, content, and documentation. There are many commercial and open source tools developed for static code analysis [3], [24]. These tools remove the unnecessary fuzz from source code and perform some automated checks to improve and ensure a certain level of quality. This can be performed very early in the development process, during this procedure the code must pass many formal tests to be considered bug free. There exist several ways of analyzing static code by exploiting the natural language found within a program's text based on compliance with different coding standards. These types of analysis may be manual, which is usually very time consuming like code inspections, or automated using one or more tools. Software Bug Prediction (SBP) considers a vital activity during software development and maintenance. SBP is a methodology related to figure out bugs in the software module by considering software metrics as a parameter [4]. Numerous studies have confirmed that machine learning techniques are suitable techniques for predicting software bug to identify defective software code [5], [6], [9]. Bug reports are basic software development tools which describe software bugs, especially in open-source software [7], [30]. To warranty the quality of software, many projects use bug reports to gather and record the bugs reported [8]. The bugs classified into two classes: intrinsic bugs refer to bugs that were introduced by one or more specific changes to the source code and extrinsic bugs refer to bugs that were introduced by changes not recorded in the version control system [5], [18]. Several techniques have been developed over the years to automatically detect bugs in source code. Often, these techniques depend on formal methods program analysis. Many studies in literature use code features as input for machine learning algorithms to perform bug prediction. The most machine learning algorithms that can be used to detect software bugs is classification techniques [10].

III. Machine Learning Techniques

Machine learning is an area of research where computer programs can learn and get better at performing specific tasks by training on historical data [2]. Machine learning algorithms can be applied to analyze data from different perspectives to allow developers to obtain useful information [10], [38]. High quantities of data are needed to develop machine learning models-based prediction [11], [31], [33]. Machine learning algorithms build models from training examples, which are then used to make predictions when faced with new examples. Supervised learning is a type of machine-learning algorithm that builds a prediction model by training the labeled data to execute the prediction task. The goal of supervised machine learning algorithms is to develop an inferring function through concluding relationships between independent variables(inputs) and dependent variables(outputs) of the training datasets [5], [27]. Classification is a method uses a data mining or machine learning approach classify the data, classification techniques deal with a software component, named classifier, this classifier invoked with inputs (features). Features are extracted from the training data examples as text, numbers, or nominal values. Bug prediction is one application of machine learning that aims to identify critical pieces in source code potential contain defects. This process can be used in software projects to earning insights into how and where bugs happen to enhance software quality.

IV. Literature Review

Software bug prediction is one of the most popular research areas in software engineering. The major aim of the software bug prediction is to detect bugs in software modules by considering software metrics as input (parameters). The research described in this paper presents a comprehensive study on machine learning techniques for software bug prediction. The following subsection covers the recent literature related to bug prediction. Considerable research has been performed on software bug prediction using machine learning techniques. For example, Wang et al. in [1] proposed a combination approach of contexts and neural network to detecting bugs. The results show that the tool can have a relative improvement up to 160% on F-score. Also, the tool can detect 48 true bugs in the list of top 100 reported bugs. Jonsson et al. in [2] evaluated automated bug assignment techniques that are based on machine learning classification. The results of study show that the prediction of accuracies is between 50% and 90% when large training sets are used. Chappell et al. in [3] presented report on using machine learning techniques for finding bugs in C programs. Hammouri et al. in [5] presented machine learning model for software bug prediction. The experiment was conducted on the basis of three supervised machine learning algorithms Naïve Bayes, Decision Tree, and Artificial Neural Networks to predict future software bugs based on historical data. The results show that the use of machine learning algorithms is effective and leads to a high rate of accuracy. The comparison results showed that the Decision Tree (DT) classifier has the best results over the others. Kumar Pandey et al. in [6] conducted compare various Bayesian network classifier and how they are useful for bugs prediction and random forest. The experimental results revealed that the Bayesian network is better than random forest. Meenakshi et al. in [7] proposed various ML models for software bug prediction. The experiment results demonstrated that the machine learning techniques are efficient and suitable approaches to predict the future software bugs and the comparison of results showed that the DT classifier has the best results over the others. Un-

Nisa Uqaili et al. in [8] proposed an approach to classify different types of bugs according to their severity and priority basis. They applied three supervised machine learning models (Naïve Bayes, Random Forest, and Multilayer Perceptron) for prediction of fault prone. The experimental results showed that the Random-Forest (RF) method better than other techniques of machine learning. Aleem et al. in [10] conducted study to a comparative the performance of some machine learning algorithms for software bug prediction. The results showed most of the applied machine learning techniques performed well on software bug prediction. Islamet et al. in [11] presented an empirical study using deep learning libraries to explore the bugs in software. They conducted 2716 comprehensive bug characteristics studies to identify the bug types and root causes of bugs. The study found that the most severe bug types in deep learning software are data bug and logic bug, where appearing more than 50% of the times and main causes of these bugs are incorrect model parameter and structural inefficiency. Sharma et al. in [13] proposed a new approach of creating a dictionary to classify critical terms and determine severity using two machine learning algorithms (Naïve Bayes Multinomial and K-nearest neighbor algorithms), and the results were evaluated based on two performance measures (accuracy and accuracy). The results demonstrated that the K-nearest neighbor classifier performs better Naïve Bayes Multinomial classifier to classify the severity of the bug Table I illustrates techniques used in previous studies on machine learning-based software bugs prediction. Bold number indicates comparative studies, capital and bold X shows the classifier giving the best results.

TABLE I. ML TECHNIQUES USED IN PREVIOUS STUDIES FOR SOFTWARE BUGS PREDICTION

Reference	Machine Learning techniques							
	DT	**NB**	**ANNs**	**RF**	**SVM**	**DL**	**K-NN**	**LR**
[1]			x					
[3]			x					
[4]		x						
[**5**]	**X**	x	x					
[6]		x						
[**7**]	**X**	x						
[**8**]		x	x	**X**				
[**10**]	x	x	**X**	x	**X**			
[11]						x		
[12]		x					x	
[13]		x						
[16]		x	x		x		x	
[**18**]		x		x	x			**X**

V. RESEARCH METHODOLOGY

The main objective of this study is to identify and analyze the latest studies that use machine learning techniques for software bug prediction. A literature review has been used as a research methodology in this study as it is a defined and methodical way of identifying, evolution, and analyzing published literature to investigate the research questions.

A. Study Selection

There are a lot of criteria to identify the relevant studies in this study and papers collected and reviewed by year of publication as it is shown in Fig. 1. For a paper to be included in this study, it must meet various inclusion criteria.

- Studies that suggest and discuss the use of machine learning techniques to predict software bugs.
- Studies that motivate and discuss the benefits of using machine learning techniques for software bug prediction.
- Studies that provide an empirical basis for the results and have been published in a high-quality journal or in conference proceedings.

B. Research Questions

This study aims to establish a starting point for future research for software bug prediction and simultaneously provide practitioners with a summary of most relevant work done in the area of software bug prediction uses machine learning techniques to heel and allow picking machine learning techniques that suits them. The research questions identified in this context are given in Table II.

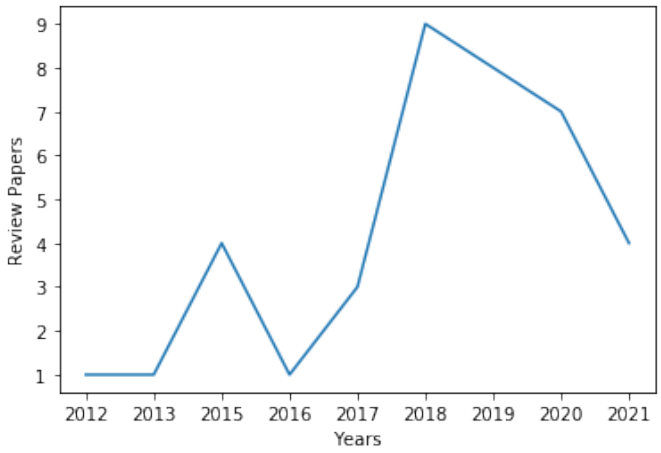

Fig. 1. Number of Papers Collected and Reviewed by Year of Publication.

TABLE II. RESEARCH QUESTIONS

RQ#	Research Question	Motivation
RQ1	Which ML models have been used for software bug prediction?	Identify the machine learning models commonly being used for software bug prediction.
RQ2	How these models have been trained and what languages have been used?	To find out how these models were trained and what languages are used.
RQ3	Which performance measures are used for software bug prediction?	Assess the performance of the machine learning techniques for software bug prediction.
RQ4	What the conclusions can we draw about the efficiency of machine learning algorithms used in predicting software bug from results presented in the selected studies?	Identify the efficiency of machine learning algorithms used in predicting software bug from results presented in the selected studies.

1) RQ1: Which ML models have been used for software bug prediction?: To answer this research question, this study identified the machine learning models commonly being used for software bug prediction in previous studies as shown in Fig. 2, and these models are:

- Decision Tree is a popular learning method used in data mining and machine learning for the purpose of regression and classification. It refers to a hierarchal model or a tree with decision nodes that have more than one branch and leaf nodes that represent the decision. Each node in a decision tree represents a feature in an instance to be classified, and each branch represents the value thresholds the contained nodes can assume. Instances are categorized beginning at the root node and sorted based on their attribute values [5], [29].
- Naïve Bayes (NB) is a supervised learning algorithm and defines as simple probabilistic classifier and efficient based on Bayes theorem with independence assumption between the features, this means that the Naive Bayes classifier is based on estimating the probabilities of the unobserved node, based on the observed probabilities [5], [22].
- Artificial Neural Networks (ANNs): ANNs are machine learning models or nonlinear classifiers used to model complex relationships between inputs and outputs for classification purposes. An ANN model contains multiple units (layers) for information processing which are known as neurons. The layers are typically named the input layer, hidden layer, and output layer [5]. When implementing a neural network, a set of consistent training values must be available to set up the expected operation of the network and a set of validation values to validate the training process [14].
- Random Forest is one of the most utilized models, due its effortlessness and the way, which it can be utilized for both characterization and relapse assignments. It is an adaptable and simple to utilize machine learning calculation, even without hyper-parameter tuning [23].
- Support Vector Machine (SVM): SVM is one of the regulated machine learning models. It is a comparatively novel learning approach used for binary classification. The primary role is to discover a hyper-plane, which divide the dimensional data completely into two categories [15], [32].
- Deep Learning (DL): DL is one of an artificial intelligence function that mimics the workings of the human brain. It allows and helps to solve complex problems with using a data set that is very diverse, unstructured, and interconnected [40].
- K-Nearest Neighbor define as a simple supervised classification algorithm in which an object is classified by looking at the K nearest objects and by choice most frequently occurring class [28].
- Logistic Regression (LR): LR is a statistical classification technique which is based on maximum likelihood estimation. It is meant for predicting the likelihood of an entity belonging one class or another class [16], [28], [37], [39].

2) RQ2: How these models have been trained and what languages have been used? To answer this research question, the essential issue of software bug prediction with machine learning techniques is how train and test the model [17]. A large and representative data set is the basis for training and testing machine learning models. So, in the literature review and in our experimental study, different and large datasets, and different programming languages such C, C++ and Java has been used to training machine learning models.

3) RQ3: Which performance measures are used for software bug prediction? To answer this research question, several measures are used for gauging the performance of different machine learning models. These performance measures are used for comparing and evaluating models developed using various machine learning techniques. A depiction of the number of studies using each performance measures is used in Fig. 3. The most used performance metric is accuracy, which is closely followed by recall, precision, and F1-score, and some less commonly metrics are H-measure, Area Under the Curve (AUC) and Receiver Operating Characteristics (ROC) curve.

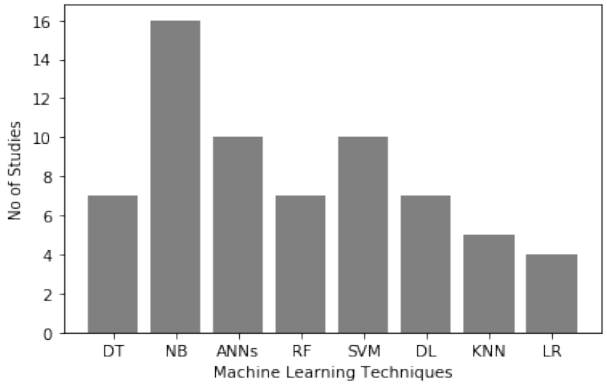

Fig. 2. Number of Studies across ML Techniques based on Classifications.

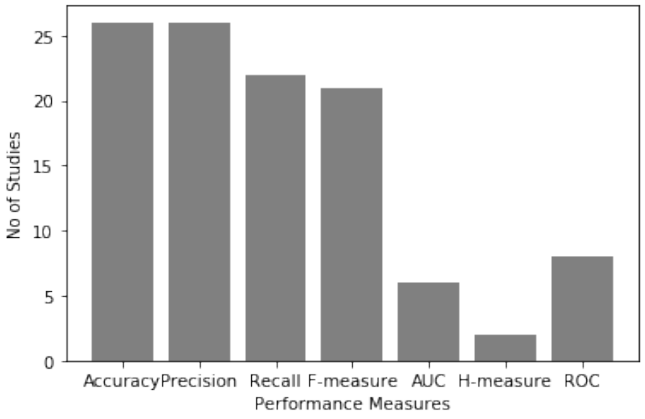

Fig. 3. Studies using different Performance Measures for SBP.

4) RQ4: What the conclusions can we draw about the efficiencies of machine learning techniques for software bug prediction from results presented in the selected studies?: To answer this research question, this study evaluates the best machine learning techniques for devolving an effective model for software bug prediction through evaluating the presented software bug prediction models in previous studies. Different machine learning techniques have different characteristic like speed, accuracy, interpretability, and simplicity. This study focused on the studies that applied machine learning algorithms and performance measures that most used. Looking at the results achieved in the literature review and the results achieved in our study, machine learning techniques are well applicable to static code analysis for software bug prediction.

VI. Software Metrics (Features) and Datasets

Software metrics are a quantitative and standard measure of some property of software that assigns numbers or symbols to attributes of the measured entity. Software metrics can be used to collect information regarding structural properties of a software design which can be further statistically analyzed, interpreted and linked to its quality. In software comprise complexity, cohesion, and coupling related metrics can be measured during the software development phases such as design or coding and it also used to calculate the quality of software [19], [34], [36]. Software metrics can be classified to static code metrics and process metrics. Static code metrics can be directly extracted from source code, like Lines of Code (LOC), Cyclomatic Complexity Number (CCN). Object oriented metrics is a subcategory of static code metrics, like Depth of Inheritance Tree (DIT), coupling between Objects (CBO), Number of Children (NOC), and Response for Class (RFC). Process metrics can be extracted from Source Code Management system based on historic changes on source code overtime. Metrics can also be classified based on development phase of software life cycle, into source code level metrics, detailed design level metrics or test level metrics. Object-oriented metrics are often used to assess the testability, maintainability or reusability of source code [20], [35]. Commonly dataset that used for software bug prediction domain is promise repository dataset. To perform this experiment, the data is obtained from the publicly available and published data in defect prediction datasets that stored software metrics along with defect information of several projects, these datasets were collected from real software projects by NASA. These public domain datasets are used in this experiment because this is a benchmarking procedure of defect prediction research [17, 21]. To perform machine learning on the available source code, it is necessary to establish a set of features that can be extracted that contain the information needed. Many studies [4, 6, 7, and 14] use software metrics as independent variables to measuring the quality of software modules and build software bug prediction models. It is intuitive to think that the bug proneness of a module is correlated with its complexity; therefore, bug prediction studies usually employ product metrics to improve prediction accuracy. The projects used in this study were developed using different programming languages and include heterogeneous code metrics like Object-Oriented (OO) metrics, Halstead metrics, Lines of Codes (LoC), and McCabe complexity. Various defects detection methods like Black box probing, automatic formal methods, etc. And different machine learning models like linear regression, the M5' model tree learner and the J48 decision tree learner have been implemented in these projects [10]. Table III, Table IV shows the information about dataset, and software metrics (features).

TABLE III. Descriptions of Datasets (Projects) used in this Study

Projects	# Modules	% Defects	Language	Description
JM1	10885	19%	C	Real-time predictive ground system: Uses simulations to generate predictions.
PC1	1107	6.8%	C	Flight software for earth orbiting satellite.
KC1	2107	15.4%	C++	Storage management for receiving and processing ground data.
KC2	523	20%	C++	Software for science data processing.

TABLE IV. Descriptions of Software Metrics (Features) used in this Study

Metrics	Type	Description
Loc	McCabe	It counts the line of code in software module.
v(g)	McCabe	Measure McCabe Cyclomatic Complexity.
ev (g)	McCabe	McCabe Essential Complexity.
iv (g)	McCabe	McCabe Design Complexity.
N	Derived Halstead	Total number of operators and operands.
V	Derived Halstead	Volume.
L	Derived Halstead	Program length.
D	Derived Halstead	Measure difficulty.
I	Derived Halstead	Measure Intelligence.
E	Derived Halstead	Measure Effort.
B	Derived Halstead	Effort estimate.
T	Derived Halstead	Time Estimator.
Locoed	Line Count	Number of lines in software module.
Locomment	Line Count	Number of comments.
Loblank	Line Count	Number of blank lines.
Locodeandcomment	Line Count	Number of codes and comments.
uniq_op	Basic Halstead	Unique operators.
uniq_opnd	Basic Halstead	Unique operands.
total_op	Basic Halstead	Total operators.
total_opnd	Basic Halstead	Total operands.
BranchCount	Branch	Total Number of branch count.

VII. Classifiers Used for Software Bug Prediction in this Study

The next step after collecting datasets is using the collected datasets to train a machine learning models potential buggy modules as it is shown is Fig. 4. Four supervised machine learning algorithms will be analyzed and evaluated in this study, which are DT, NB, RF and LR. These algorithms were chosen because are the most algorithms used in previous studies.

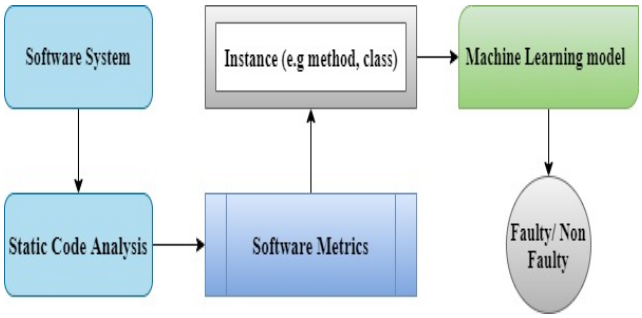

Fig. 4. Structure of Software Bug Prediction Model.

VIII. Building and Evaluation of Prediction Models

Most studies of software bug prediction divide the data into two sets: a training set and a test set. The training set is used to train the bug prediction models, whereas the testing set is used to evaluate the performance of the bug prediction models. After building the prediction model, we need to evaluate the performance of the model. To evaluate the performance of using machine learning models in software bug prediction in this study used a set of performance measures based on the confusion matrixes and ROC (Receiver Operating Characteristic) Curves. Confusion matrix(correlation matrix) is often used to describe the performance of machine learning models(classification methods) using a set of test data, correlation summarizes the results of the testing algorithm and provides a report of (1) True Positives (TP), (2) False Positives (FP), (3) True Negatives (TN), and (4) False Negatives (FN). ROC curves are plots the false positive rate on the x-axis and true positive rate on the y-axis over all possible or potential classification thresholds. The subsections bellow describes the confusion matrix and performance measures applied as it is shown in Table V and equations.

- Accuracy: Accuracy is the ratio of true results that calculated as the sum total of true positive and true negative instances divided by 100. The top (maximum) accuracy is 1, whereas the low (minimum) accuracy is 0. Accuracy can be computed by using the following formula:

$$\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{TN} + \text{FP} + \text{FN})} \quad (1)$$

- Precision: Precision is defined as the number of true positive predictions divided by the total number of positive predictions or fraction of true positive and predicted yes instances. The top (maximum) precision is 1, whereas the low (minimum) is 0 and it can be calculated as:

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})} \quad (2)$$

- Recall: Recall is the number of positive predictions divided by the total number of positives or defined as the fraction between true positive instances and actual yes instances. The top (maximum) recall is 1, whereas the low (minimum) is 0. The formula of recall given below:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

- F1-score: F1-score is weighted harmonic mean of precision and recall or defined as the fraction between product of the recall and precision to the summation of recall and precision parameter of classification, it is used to combine the recall and precision measures in one measure to compare different machine learning algorithms. F1-score formula is given below:

$$\text{F1} - \text{score} = \frac{(2 * \text{Recall} * \text{Precision})}{(\text{Recall} + \text{Precision})} \quad (4)$$

TABLE V. The Correlation Matrix

Predicted	**Actual**	
	Class X	Class Y
Class X	TN	FP
Class Y	FN	TP

IX. Result and Discussion

This study aimed at improving the understanding of the process of software bug prediction especially using supervised machine learning techniques. In the literature review, several papers were found that discussed machine learning models for predicting software bugs that classify the defective and non-defective module. It was observed in the RQ1 analysis that most of the machine learning techniques used in software bug prediction are NB, ANNs, and SVM. As it is noted in the RQ2 analysis that studies used different performance measures. The experiment of this study was performed in PYTHON environment to evaluate four machine learning algorithms: DT, NB, RF and LR. The evaluation process is implemented with real datasets. Experimental results are collected and evaluated based on various performance measures (accuracy, precision, recall, F1-score and ROC Curves). Results demonstrated that the machine learning algorithms are efficient approaches to predict software bugs. The comparison results demonstrated that the Decision Tree (DT) and Random Forest (RF) classifiers have the best results. Tables VI to IX show the performance of proposed models on the four data sets based on all performance measures. The maximum (best) accuracy value is 99%, which was achieved by Decision Tree (DT) and Random Forest (RF) models in JM1, PC1and KC1 datasets. The maximum (best) precision value is 99%, which was achieved by Decision Tree (DT) and Random Forest (RF) models in JM1, PC1and KC1 datasets. The maximum (best) recall value is 100%, which was achieved by Decision Tree (DT) and Random Forest (RF) models in all datasets. The maximum (best) F1-score value is 99%, which was achieved by Decision Tree (DT) and Random Forest (RF) models in

PC1 dataset. The average accuracy of the proposed models on the four data sets is shown in Fig. 5 and Fig. 6. As shown, the two ML models Decision Tree (DT) and Random Forest (RF) achieved a high average accuracy rate. The average value for the accuracy rate in all datasets for the two models is over 98.5% on average. The minimum value appears for Naive Bayes (NB) model in the JM1 dataset, because the data set is small and the Naive Bayes (NB) model needs a large data set in order to achieve a high accuracy value. Fig. 7 to Fig. 10 presents the ROC Curves of proposed models on the four data sets. The results show that Decision Tree (DT) and Random Forest (RF) models have better values than Naive Bayes (NB) and Logistic Regression (LR) models. For evaluating the effectiveness of the proposed models, in Tables X and XI we have compared the results of our study with the results of three others studies [4, 7, and 10] which used the same dataset and different performance measures (Accuracy, Precision, Recall, and F1-score). The results showed that our proposed models performed better than others models. After a comprehensive study of Machine Learning techniques, there must be a deterministic strategy for selecting machine learning techniques to predict software bugs.

TABLE VI. PERFORMANCE MEASURES OF THE PROPOSED MODELS OVER JM1 DATASET

proposed model	Performance measures			
	Accuracy	**Precision**	**Recall**	**F1-score**
DT	**0.99**	**0.99**	**1.00**	**0.99**
NB	0.80	0.81	0.97	0.89
RF	**0.99**	**0.99**	**1.00**	**0.99**
LR	0.81	0.82	0.99	0.89

TABLE VII. PERFORMANCE MEASURES OF THE PROPOSED MODELS OVER PC1 DATASET

proposed model	Performance measures			
	Accuracy	**Precision**	**Recall**	**F1-score**
DT	**0.99**	**0.99**	**1.00**	**1.00**
NB	0.91	0.94	0.96	0.95
RF	**0.99**	**0.99**	**1.00**	**1.00**
LR	0.93	0.94	0.99	0.96

TABLE VIII. PERFORMANCE MEASURES OF THE PROPOSED MODELS OVER KC1 DATASET

proposed model	Performance measures			
	Accuracy	**Precision**	**Recall**	**F1-score**
DT	**0.99**	**0.99**	**1.00**	**0.99**
NB	0.85	0.88	0.96	0.92
RF	**0.99**	**0.99**	**1.00**	**0.99**
LR	0.85	0.87	0.96	0.92

TABLE IX. PERFORMANCE MEASURES OF THE PROPOSED MODELS OVER KC2 DATASET

proposed model	Performance measures			
	Accuracy	**Precision**	**Recall**	**F1-score**
DT	**0.98**	**0.98**	**1.00**	**0.99**
NB	0.83	0.83	0.98	0.90
RF	**0.98**	**0.98**	**1.00**	**0.99**
LR	0.84	0.86	0.96	0.91

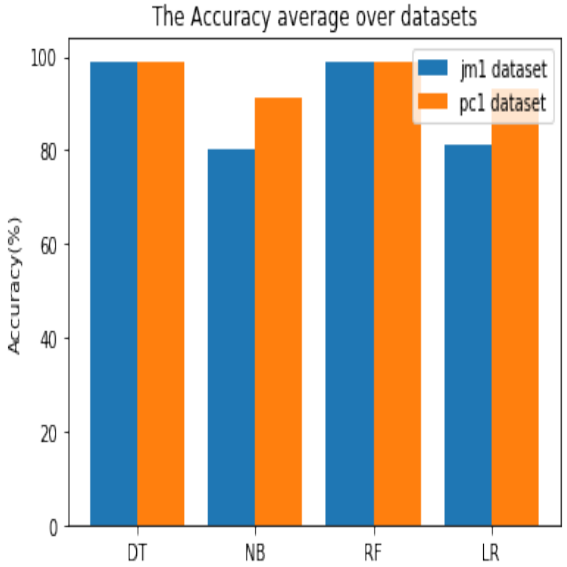

Fig. 5. Average of Accuracy Measure of Models across the JM1 and PC1 Dataset.

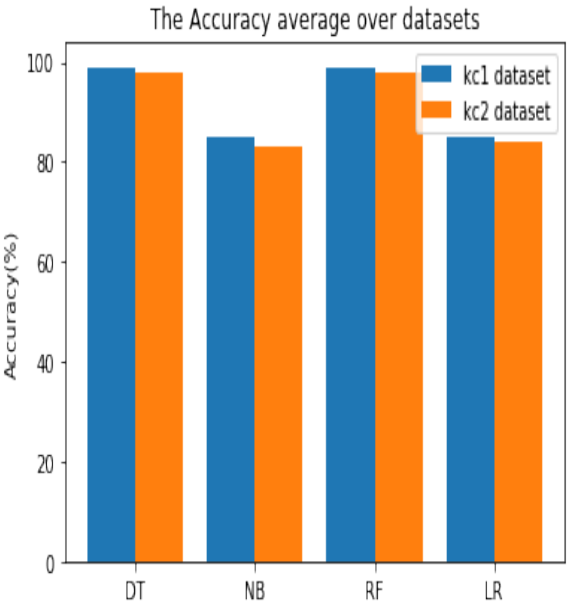

Fig. 6. Average of Accuracy Measure of Models across the KC1 and KC2 Dataset.

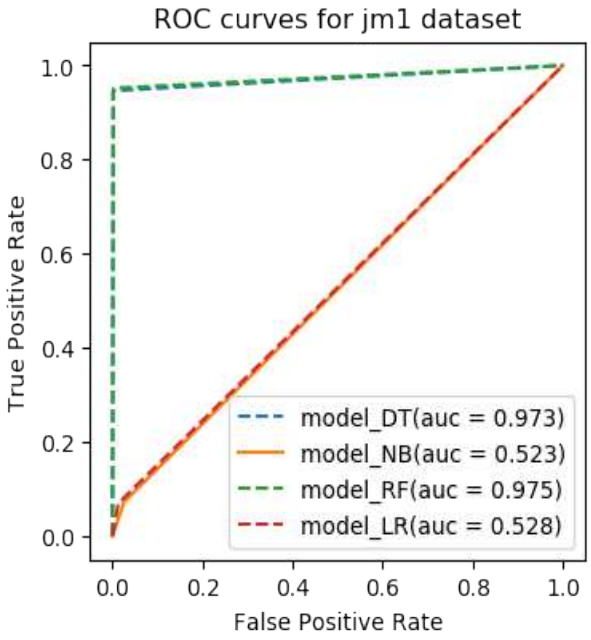

Fig. 7. Comparison of ROC Curves for Models across the JM1 Dataset.

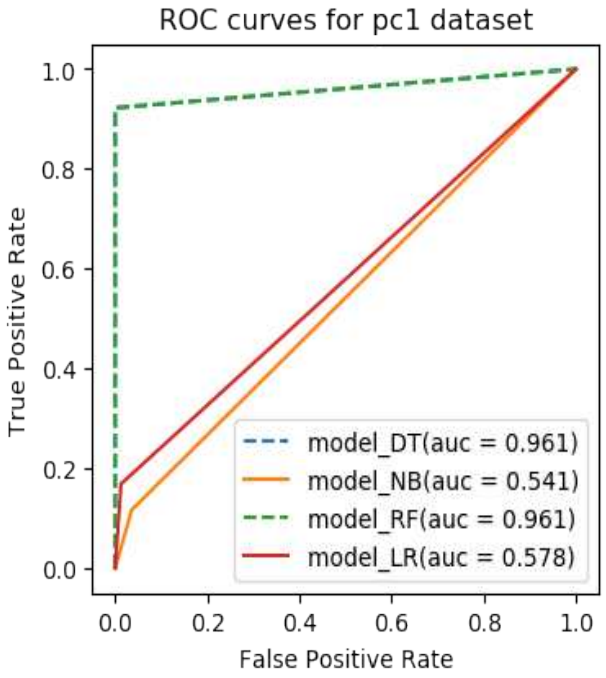

Fig. 8. Comparison of ROC Curves for Models across the PC1 Dataset.

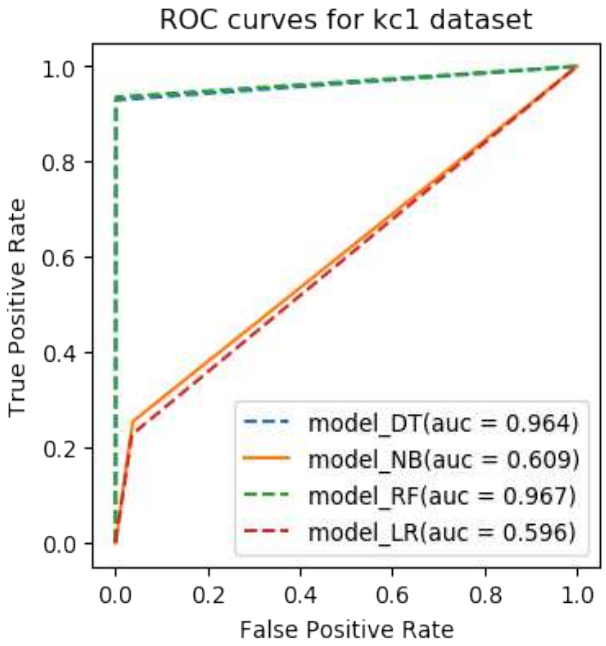

Fig. 9. Comparison of ROC Curves for Models across the KC1 Dataset.

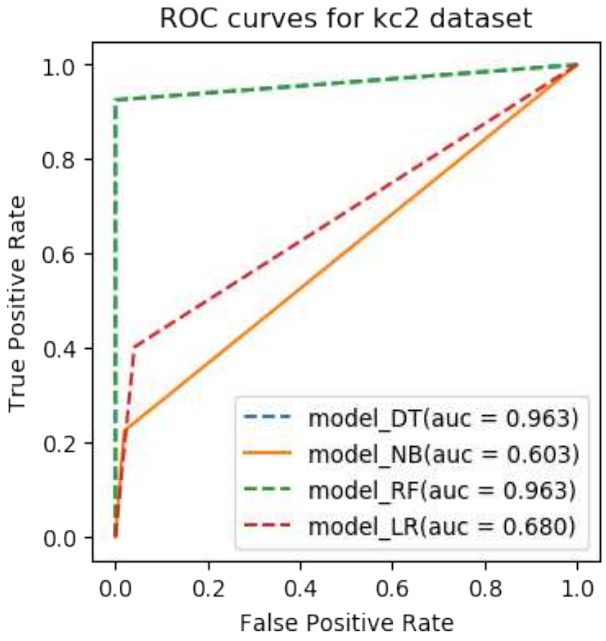

Fig. 10. Comparison of ROC Curves for Models across the KC2 Dataset.

TABLE X. COMPARING THE RESULTS OF OUR STUDY WITH THE RESULTS OF STUDIES WHICH USES THE SAME DATASET AND ALGORITHMS ACROSS THE JM1 AND PC1 DATASET

JM1 dataset					
Performance measure	**ML models**	**Studies**			
		First Study	**Second Study**	**Third Study**	**Our study**
Accuracy	DT	-	-	0.81	**0.99**
	NB	-	-	0.81	**0.80**
	RF	-	-	0.82	**0.99**
F1-score	DT	-	-	0.90	**0.99**
	NB	0.75	-	0.89	**0.89**
	RF	0.76	-	0.90	**0.99**
	LR	0.74	-	-	**0.89**
pc1 dataset					
Accuracy	DT	-	-	0.93	**0.99**
	NB	-	-	0.88	**0.91**
	RF	-	-	0.93	**0.99**
F1-score	DT	-	-	0.97	**1.00**
	NB	0.89	-	0.94	**0.95**
	RF	0.91	-	0.97	**1.00**
	LR	0.91	-	-	**0.96**

TABLE XI. COMPARING THE RESULTS OF OUR STUDY WITH THE RESULTS OF STUDIES WHICH USES THE SAME DATASET AND ALGORITHMS ACROSS THE KC1 AND KC2 DATASET

kc1 dataset					
Performance measure	**ML models**	**Studies**			
		First Study	**Second Study**	**Third Study**	**Our study**
Accuracy	DT	-	-	0.84	**0.99**
	NB	-	0.82	0.82	**0.85**
	RF	-	-	0.85	**0.99**
Precision	NB	-	0.80	-	**0.88**
Recall	NB	-	0.83	-	**0.96**
F1-score	DT	-	-	0.92	**0.99**
	NB	0.82	0.81	0.90	**0.92**
	RF	0.82	-	0.92	**0.99**
	LR	0.81	-	-	**0.92**
kc2 dataset					
Accuracy	DT	-	-	0.82	**0.98**
	NB	-	-	0.84	**0.83**
	RF	-	-	0.82	**0.98**
F1-score	DT	-	-	0.89	**0.99**
	NB	0.80	-	0.90	**0.90**
	RF	0.76	-	0.89	**0.99**
	LR	0.79	-	-	**0.91**

X. CONCLUSION

Software bug prediction is very important field in static code analysis to improve software quality and reliability. It is an approach, in which a prediction model is constructed for the purpose of predicting future software defects based on historical data using some software metrics. Many approaches have been presented using various datasets, various metrics, and various performance measures. The aims of this study are successfully achieved. The aims are evaluate and present comprehensive study on machine learning techniques have been used for software bug prediction in recent years and apply the best techniques for software bug prediction in this study. To compare and evaluate the performance of the proposed models, we used different performance measures. The results concluded that ML techniques are gaining interest in software bug prediction, to improve the efficiency of bug detection. Four NASA public datasets were chosen for this experiment and analyze the performance of models. The experimental results revealed that the DT and RF classifiers are better than others classifiers. Static code analysis requires further research to identify and detect of software bugs and several machine learning techniques can be used to improve results. As a future work, we plan to introduce other machine learning techniques with data balancing techniques to improve the accuracy for predicting software bugs.

ACKNOWLEDGMENT

The authors gratefully acknowledge the financial assistance from the Institute of Information Science, Faculty of Mechanical Engineering and Informatics, University of Miskolc.

REFERENCES

[1] Y. Li, S. Wang, T. N. Nguyen, and S. V. Nguyen, “Improving bug detection via context-based code representation learning and attention-based neural networks”, in Proceedings of the ACM on Programming Languages, vol. 3, OOPSLA, paper no. 162, pages 1–30, 2019.

[2] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, “Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts”, Empirical Software Engineering, vol. 21, pp. 1533–1578, 2016.

[3] T. Chappelly, C. Cifuentes, P. Krishnan and S. Gevay, “Machine learning for finding bugs: An initial report” in IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation, Klagenfurt, Austria, 21 -21 February 2017, pp. 21–26.

[4] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, “BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques”, Expert Systems with Applications, vol. 144, paper no. 113085, 2020.

[5] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, “Software bug prediction using machine learning approach”, International Journal of Advanced Computer Science and Applications, vol. 9, no. 2, pp. 78–83, 2018.

[6] S. K. Pandey, R. B. Mishra, and A. K. Triphathi, “Software bug prediction prototype using Bayesian network classifier: A comprehensive model”, Procedia Computer Science, vol. 132, pp. 1412–1421, 2018.

[7] S. S. Meenakshi, “Software bug prediction using machine learning approach”, International Research Journal of Engineering and Technology, vol. 6, no. 4, pp. 4968–4971, 2019.

[8] I. U. N. Uqaili, S. N. Ahsan, “Machine learning based prediction of complex bugs in source code”, The International Arab Journal of Information Technology, vol. 17, no. 1, pp. 26–37, 2020.

[9] Károly, Nehéz, and Khleel Nasraldeen Alnor Adam. "Tools, processes and factors influencing of code review." Multidiszciplináris Tudományok 10.3 (2020): 277-284.

[10] Aleem, Saiqa, Luiz Fernando Capretz, and Faheem Ahmed. "Comparative performance analysis of machine learning techniques for software bug detection." ITCS, CST, JSE, SIP, ARIA, DMS (2015): 71-79.

[11] M. J. Islam, P. Pan, G. Nguyen, and H. Rajan,"A comprehensive study on deep learning bug characteristics", in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Tallinn, Estonia, 26–30 August 2019, pages 1–11, 2019.

[12] S. Gitika, S. Sharma, and S. Gujral. "A novel way of assessing software bug severity using dictionary of critical terms", Procedia Computer Science, vol. 70, pp. 632–639, 2015.

[13] P. Maltare and V. Sharma, "Implementation advance technique for prediction bug using machine learning", International Journal of Computer Science and Information Technologies, vol. 8, no. 1, pp. 16–19, 2017.

[14] S. D. Immaculate, M. F. Begam, and M. Floramary. "Software bug prediction using supervised machine learning algorithms", in International Conference on Data Science and Communication, Bangalore, India, 1-2 March 2019, pages 1–7, 2019.

[15] G. Rodríguez-Pérez, A. Serebrenik, A. Zaidman, D. M. Germán and J. M. Gonzalez-Barahona, "How bugs are born: a model to identify how bugs are introduced in software components", Empirical Software Engineering, vol. 25, pp. 1294–1340, 2020.

[16] M. Sharma, P. Bedi, K.K. Chaturvedi, and V.B. Singh, "Predicting the priority of a reported bug using machine learning techniques and cross project validation", in 12th International Conference on Intelligent Systems Design and Applications, Kochi, India, 27-29 November 2012, pp. 539–545, 2012.

[17] Shirabad, J. Sayyad, and Tim J. Menzies. "The PROMISE repository of software engineering databases." School of Information Technology and Engineering, University of Ottawa, Canada 24 (2005).

[18] M. Efendioglu, A. Sen, and Y. Koroglu. "Bug prediction of system C models using machine learning", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 3, pp. 419–429, 2019.

[19] Rajkumar, V. and V. Venkatesh. "Hybrid Approach for Fault Prediction in Object-Oriented Systems." (2017).

[20] Meiliana, Syaeful Karim, et al. "Software Metrics for Fault Prediction Using Machine Learning Approaches." IEEE (2017).

[21] Iqbal, Ahmed, et al. "Performance analysis of machine learning techniques on software defect prediction using NASA datasets." Int. J. Adv. Comput. Sci. Appl 10.5 (2019): 300-308.

[22] Baarah, Aladdin, et al. "Machine learning approaches for predicting the severity level of software bug reports in closed source projects." Mach Learn (2019).

[23] Kukkar, Ashima, et al. "A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting." Sensors 19.13 (2019): 2964.

[24] Moustafa, Sammar, et al. "Software bug prediction using weighted majority voting techniques." Alexandria engineering journal 57.4 (2018): 2763-2774.

[25] ÖZTÜRK, Elife, Kökten Ulaş Birant, and Derya Birant. "An Ordinal Classification Approach for Software Bug Prediction." Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi 21.62 (2019): 533-544.

[26] Ferenc, Rudolf, et al. "An automatically created novel bug dataset and its validation in bug prediction." Journal of Systems and Software 169 (2020): 110691.

[27] Pecorelli, Fabiano, and Dario Di Nucci. "Adaptive selection of classifiers for bug prediction: A large-scale empirical analysis of its performances and a benchmark study." Science of Computer Programming 205 (2021): 102611.

[28] Sharma, Shubham, and Sandeep Kumar. "Analysis of Ensemble Models for Aging Related Bug Prediction in Software Systems." ICSOFT. 2018.

[29] Kumar, Raj. "Multiclass Software Bug Severity Classification using Decision Tree, Naive Bayes and Bagging." Turkish Journal of Computer and Mathematics Education (TURCOMAT) 12.2 (2021): 1859-1865.

[30] Ferenc, Rudolf, et al. "Deep learning in static, metric-based bug prediction." Array 6 (2020): 100021.

[31] Ye, Xin, et al. "Bug Report Classification using LSTM architecture for more accurate software defect locating." 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2018.

[32] Bani-Salameh, Hani, and Mohammed Sallam. "A Deep-Learning-Based Bug Priority Prediction Using RNN-LSTM Neural Networks." e-Informatica Software Engineering Journal 15.1 (2021).

[33] Pascarella, Luca, Fabio Palomba, and Alberto Bacchelli. "Re-evaluating method-level bug prediction." 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018.

[34] Puranik, Shruthi, Pranav Deshpande, and K. Chandrasekaran. "A novel machine learning approach for bug prediction." Procedia Computer Science 93 (2016): 924-930.

[35] Saharudin, S. N., Wei, K. T. & Na, K. S. (2020). Machine Learning Techniques for Software Bug Prediction: A Systematic Review. Journal of Computer Science, 16(11), 1558-1569.

[36] Gupta, Varuna, N. Ganeshan, and Tarun K. Singhal. "Developing software bug prediction models using various software metrics as the bug indicators." International Journal of Advanced Computer Science and Applications (IJACSA) 6.2 (2015).

[37] Baarah, Aladdin, et al. "Machine learning approaches for predicting the severity level of software bug reports in closed source projects." International Journal of Advanced Computer Science and Applications 10.10.14569 (2019).

[38] Qin, Fangyun, Xiaohui Wan, and Beibei Yin. "An empirical study of factors affecting cross-project aging-related bug prediction with TLAP." Software Quality Journal 28.1 (2020): 107-134.

[39] Qin, Fangyun, et al. "Studying aging-related bug prediction using cross-project models." IEEE Transactions on Reliability 68.3 (2018): 1134-1153.

[40] Som Gupta and Sanjai Kumar Gupta, "A Systematic Study of Duplicate Bug Report Detection" International Journal of Advanced Computer Science and Applications(IJACSA), 12(1), 2021.