A Comparative Analysis for Machine Learning based Software Defect Prediction Systems

Murat Cetiner
Department of Computer Engineering
Istanbul Kultur Univercity
Istanbul, Turkey
murat_cetiner_@hotmail.com

Ozgur Koray Sahingoz
Department of Computer Engineering
Istanbul Kultur Univercity
Istanbul, Turkey
sahingoz@gmail.com

***Abstract*—In the Software Engineering concept, the prediction of the software defects plays a vital role in increasing the quality of the software systems, which is one of the most critical and expensive phases of the software development lifecycle. While the use of software systems is increasing in our daily lives, their dependencies and complexities are also increasing, and this results in a suitable environment for defects. Due to the existence of software defects, the software produces incorrect results and behaviors. What is more critical than defects, is finding them before they occur. Therefore detection (and also prediction) of the software defects enables the managers of the software to make an efficient allocation of the resources for the maintenance and testing phases. In the literature, there are different proposals for the prediction of software defects. In this paper, we made a comparative analysis about the machine learning-based software defect prediction systems by comparing 10 learning algorithms like Decision Tree, Naive Bayes, K-Nearest Neighbor, Support Vector Machine, Random Forest, Extra Trees, Adaboost, Gradient Boosting, Bagging, and Multi-Layer Perceptron, on the public datasets CM1, KC1, KC2, JM1, and PC1 from the PROMISE warehouse. The experimental results showed that proposed models result in proper accuracy levels for software defect prediction to increase the quality of the software.**

Keywords—software defect prediction, machine learning, classification, classification with python

I. INTRODUCTION

Today, due to the use of increasingly sophisticated software systems and their dependencies, increasing the quality of the software, in terms of meeting the functional and non-functional requirements, has become an attractive workspace that is needed to be considered with growing importance. For expressing software quality, ISO 25010 standards can be evaluated in terms of quality class categories such as software quality assurance activities and fundamental quality of the software product: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability as detailed in Fig. 1. As seen from this figure, functional suitability which requires that produced software product must meet the needs and requirements of its users. In this concept, the functional completeness, correctness, and appropriateness are very crucial, which are directly related to error-free or at least minimized error containing software.

Additionally, in a high-quality software development process, the requirement analysis phase needs to be followed by design, coding, application, testing and integration, maintenance, and support phases [2]. Software errors can occur in any part of these phases; they need to be detected before they happen in the real execution. Software quality assurance activities and operations can be used to identify possible faulty software modules, and they allow the software developers/testers to detect and track potential software flaws for helping them to develop even higher quality software systems. There may also be some predictions about whether the modules in the next version of the software will be defective or not.

Error detection and error correction are essential parts of the software development process. Errors in software systems cause not meeting of the software requirements, conditions, and some expectations of the users. Typically, software systems tend to contain several errors which are caused by arithmetic, logic, syntax, or teamwork. Defining and correcting these software errors is often a long process that takes a lot of time for detection. Software systems can take on crucial tasks in everyday life, especially in communities that use computer systems. For this reason, software errors can cause significant problems in small or severe sizes.

With regards to the current technological developments various methods are used for detection and minimization of the software defects. Depending on their types, they can require more resources like time and money to analyze the software implementations. Machine learning-based techniques are one of the most popular ones, which need a dataset for training. After 2005, machine learning algorithms appeared in large numbers, the related articles of other methods have decreased. Until 2018, the number of articles using machine learning algorithms exceeds 90% of all related

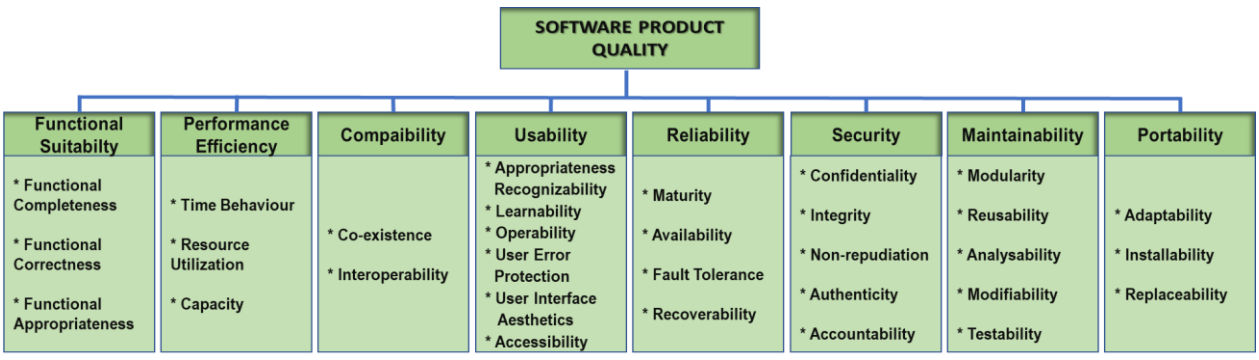

Fig. 1. Software Product Quality Model Defined in ISO/IEC 25010 [1]

articles [3]. For making a proper comparison with the literature and showing the ability of the proposed systems generally, public datasets are preferred. However, traditionally these datasets are unbalanced due to real-world processes. Software defect prediction systems need to be used to include an unbalanced class distribution in a structure where the number of records for different classes is not equal in its contents due to the structure of the datasets surveyed.

A software defect prediction method is an approach that helps in software testing and software maintenance lifecycles. Software error means that software, in general, cannot be used as aimed. If we expand the definition a little further, software defects result in the software system does not meet the functional requirements due to some type of errors. According to standard 104 of the Institute of Electrical and Electronics Engineers (IEEE) [24], there are different types of classifications for software errors.

- *Defect* is the difference between the expected and actual results of the software systems. It can sometimes easily be resolved, and sometimes can cause very vital problems and needs very much extra time to find and resolve. The programmers who implement the designed processes are generally main responsible of these types of errors.
- *Error* is a mistake of the humans, which lead the defects and bugs.
- *Failure* is encountered if the software with a defect is executed, the system might crash or fail to depend on the current values of the variables, and this results in failure.

The rest of the paper is organized as follows. The literature review is done in the next Section. The proposed system and its design details are explained in Section 3. Experimental results are depicted and compared in Section 4, and finally, conclusions and future works are drawn.

II. Related Work

The software can be implemented not only for the standard desktop computers but also for small computing devices such as sensor networks [12]. In the literature, there are many works focused on the detection of software defects by using different, especially by using various machine learning models. In [4], Akba et al. allocated software modules to subsets to estimate errors based on the size of the number of rows and have proposed an estimated model using the C4.5 Decision Tree (DT) based on minimum, maximum, total, and average values at the class level in each subset.

Pelayo and Dick, [5] applied the Smote algorithm to datasets used in software defect prediction simultaneously to increase the minority class and reduce the majority class with different percentage values. They measured the success of the C4.5 DT algorithm's G-average classification. In this way, they showed that the success of classification is increased. In this study, the majority of class data was extracted from the dataset. In our research, the current majority class was replicated, rather than reducing the class. New dataset software metrics obtained in this way examined the data gain for information gain and investigated the impact of algorithms such as C4.5 and PART.

In [6], authors tested supervised and unsupervised machine learning algorithms on the NASA dataset. 11 methods were tested on 15 datasets, comparing F-criteria, absolute error, and accuracy. Accordingly, Support Vector Machine (SVM), Multi-layered Artificial Neural Networks, and Sampling (Bagging) learning methods by putting it in place are good. Wang et al. [7] were used unbalanced learning methods for software defect prediction datasets that are unbalanced; they examined threshold value handling and union algorithms. In the study, they proposed a unity learning model that will determine the appropriate parameter values and percentages to stabilize the data. Accordingly, Adaboost.NC method was shown to be good. In this study, the Smote algorithm was used in a hybrid manner with the booster learning method (SmoteBoost), and the results were compared with the Naive Bayes (NB) and Random Forest (RF) algorithms.

Prasad et al. [8] presented research on data mining and machine learning methods with different software metrics found in the literature. The purpose of the study is to create a study. To help software developers uncover the effects of existing different software metrics, software failure, and develop better quality software. The successful metrics and machine learning methods mentioned in this study have a positive effect on us in terms of revealing the purpose of the work we do in this article. In [9], the authors focused on unbalanced software defect prediction datasets. The cost of classing the faulty class is higher than in the inert class. So, considering the unbalanced class problem, the weighted smallest squares suggested the twin SVM. They used g-average and F-criteria as a performance criterion in the study. A classification approach has been developed to balance the cost of being inaccurate and not being inaccurate, which is costly in terms of classification. In this study, datasets with balanced class distribution have been shown to yield more successful results in software defect prediction studies. The difference in our study is balanced by the dataset Smote algorithm, and the information gain has been examined and improved in terms of earnings ratio. In general, literature studies show that software defect prediction studies are a popular and highly studied subject, and studies often use Halstead, McCabe method, and level software metrics and NASA software datasets that are open to all. In these studies, it is seen that the performance of rule-based classifiers and DT comes to the forefront, and it is seen that a study is necessary to address the shortcomings and problems of these approaches. In our research, software datasets that are balanced by the Smote algorithm examined the impact of DT and rule-based classifiers on classification and increased its success.

In [10], Batista et al. conducted tests measuring learning performance on 13 datasets by increasing synthetic data and reducing synthetic data through methods called “Smote + Tomek” and “Smote + ENN”. In the overall results obtained, synthetic data increase methods were superior to synthetic data reduction methods. Lessmann et al. [11] have compared classification success using 22 classers and 10 NASA-generated publicly generated public software bug prediction datasets for software defect prediction. They stated that the results yielded successful results by metric-based classifiers, but there were no significant differences in comparison in terms of the success of the classes. In our study, the data preprocessing process was applied to a dataset with the Smote algorithm.

In [13], to compare their effectiveness in the prediction process on software error estimation, two comparisons were made for the accuracy of the proposed approach, in which five

TABLE I. DATASET DESCRIPTION

Name	*Language*	*Information*	*#Modules*	*#Attributes*	*#Defects*	*#Non-defects*	*Defect Ratio*
CM1	C	NASA spacecraft instrument	498	22	49	449	9.83
KC1	C++	Storage management for receiving and processing ground data	2109	22	326	1783	15.45
KC2	C++	Storage management for receiving and processing ground data	522	22	105	415	20.50
JM1	C	Real-time predictive ground system	10885	22	2106	8779	19.35
PC1	C	Flight software for Earth orbiting satellite	1109	22	77	1032	6.94

classifiers were tested. The three-feature selection meta-intuitive algorithm was used to make the most useful metrics from the original ones and then present them to the classifiers to estimate using only those metrics. All of them proved their efficiency with the Random Forest classifier.

III. IMPLEMENTATION

As mentioned before, the detection of whether a defect exists in the software is a crucial thing for increasing the quality of the software and also an efficient allocation of the resources to the maintenance and testing phases of the software lifecycle. This study was carried out using machine learning algorithms and deep learning algorithms along with inferences made from related studies [22, 23]. In this Section, primarily, the error definitions are emphasized. Then datasets and metrics used are explained. In the last part, the method used is specified.

A. Dataset

The dataset is a structure where raw data related to a subject is kept. Public software defect prediction dataset repositories are increasing day by day. One of them is the PROMISE dataset, owned by NASA, which is a space researcher, and in this study, CM1, KC1, KC2, JM1 and PC1 datasets were used in the PROMISE datasets [14].

These datasets' properties are given in Table 1. The datasets used have some measurement values and variables. Each record of the dataset contains a class label that knows in the relevant software module or expresses the state of the backward reported defect or non-defect state. The existence of the error indicates that changes or updates must be made in the relevant software module or other associated modules [15].

The source dates for all datasets are December 2, 2004 and the Donor is Tim Menzies. All datasets have none missing attributes [14].

B. Metrics

All these metrics are shown in Table 2 with their descriptions.

The dataset used in the study consist of McCabe metrics, Halstead metrics, and other metrics. Although some of the method-level metrics were proposed by Halstead and McCabe in the 1970s, they are still popularly used today.

The 4 McCabe metrics used in this study correspond to the method level. Method-level metrics are metrics that focus on programming concepts. These metrics are easy to collect from object-based source code, and it is determined that the modules that are prone to error are likely to be errors during the system test or field tests [16].

The 11 Halstead metrics used in the study, on the other hand, are metrics with numerical values that are mostly related to the program. These metrics are easily accessible in all software. The defects metric is a Boolean value. All other metrics are numerical [14].

TABLE II. DATASET METRICS

Metric	*Description*
loc	McCabe's line count of code
v(g)	McCabe "cyclomatic complexity"
ev(g)	McCabe "essential complexity"
iv(g)	McCabe "design complexity"
n	Halstead total operators + operands
v	Halstead "volume"
l	Halstead "program length"
d	Halstead "difficulty"
i	Halstead "intelligence"
e	Halstead "effort"
b	Halstead "delivered bugs"
t	Halstead's time estimator
lOCode	Halstead's line count
lOComment	Halstead's count of lines of comments
lOBlank	Halstead's count of blank lines
lOCodeAndComment	Lines of code and comments
uniq_Op	Unique operators
uniq_Opnd	Unique operands
total_Op	Total operators
total_Opnd	Total operands
branchCount	Of the flow graph
defects	The module has/has not one or more reported defects

C. Use Of Machine Learning Techniques

In this study, the comparative general performance analysis of different machine learning techniques for software defect prediction was investigated. Machine learning techniques have proven to be useful for software defect prediction. The data obtained from the software store contains a lot of information in the evaluation of the software quality. Thanks to this information, it is easier to find software defects along with machine learning techniques. Machine learning techniques fall into two broad categories to compare their performance: supervised learning and unsupervised learning [6], shown in Fig. 2.

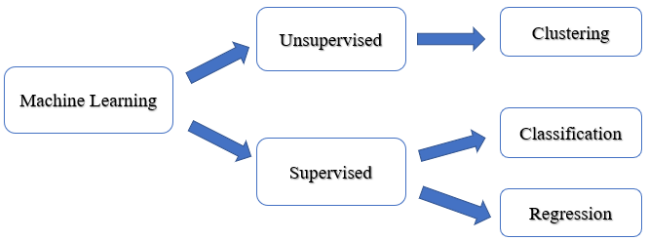

Fig. 2. Machine Learning Techniques

Supervised machine learning creates a model that makes evidence-based predictions within uncertainty. A controlled learning algorithm receives a known set of input data and known responses to the data, then trains a model to create reasonable predictions for responding to new data. If you have known data for the output you are trying to guess; you can use supervised learning. Supervised learning uses classification and regression techniques to develop predictive models. Common algorithms for classification include SVM, DT, K-Nearest Neighbor (KNN), NB, Discriminant Analysis, Logistic Regression, and Neural Networks (NN). Common regression algorithms include the Linear Model, the nonlinear Model, Regulation, DT, NN. Machine learning algorithms used in this study are as follows;

1) Decision Tree: The decision node has basic components such as branches and leaves. The input area inside a DT is divided into intersensory regions, and each region is given a value or label to determine the data points. The mechanism of the DT is transparent, and the DT structure can be monitored. Most DT consist of two stages. In the first stage, a huge tree is built and then pruned the tree in the second step to avoid the problem of adding. It is then used for pruning tree classification purposes [6].

2) Naive Bayes: NB classification algorithm is a classification algorithm named after Mathematician Thomas Bayes. Aims to determine the class of data presented to the system through a series of calculations defined by probability principles. In the NB classification, the system provides a certain amount of data. There must be a class of data offered for teaching. With probability operations on the taught data, the new test data presented to the system is operated according to the probability values obtained before, and the given test data is determined in which category. The more data taught, the more precise it can be to determine the actual category of test data [17].

3) K-Nearest Neighbor: In the classification process, the k value determines the number of elements to look at. The algorithm contains a training data. When a new value arrives, distances are calculated to the k value, and the new value is added to a set. In the distance calculation process, distance calculation methods such as Euclid distance and Manhattan distance can be used. The algorithm consists of five steps. The k value is determined first. The Euclid distances from other objects to the target object are calculated. Distances are sorted, and the nearest neighbors are located, depending on the minimum distance. The nearest neighbor categories are aggregated. The most appropriate neighbor category is selected. Needs to have a large training set and select the k value appropriately [17].

4) Support Vector Machine: SVM is mainly used to separate data from two classes in the most appropriate way. For this, the decision limits or, in other words, determine hyperplanes. It's effective in high-dimensional space. It is effective when the number of dimensions is greater than the number of samples. For the decision to be resistant to the new data, the boundary line must be closest to the boundary lines of the two classes. The points closest to this boundary line are called support points [8].

5) Random Forest Classifier: RF Algorithm is an algorithm that aims to increase the classification value by producing multiple DT during the classification process. Individually created DT together to form a forest of decision. The DT here is randomly selected subsets from the data set to which they are attached [18]. It results in a very short time. In this study, the number of trees was selected as 100.

6) Extra Trees Classifier (ET): When you grow a tree in a RF, it is considered to divide a random subset of properties on each node. Instead of searching for the best possible thresholds, it is possible to make trees even more random using random thresholds for each property. It is simply called the highly randomized trees community. Once again, it makes more bias value for a low variant. Finding the best possible threshold for each property on each node is one of the tasks that take the most time to grow trees [19].

7) Adaboost Classifier (AC): A predictor said one way to correct the pre-predictor is that the pre-predictor pays more attention to the missing education data. This method is used in the Adaboost algorithm. To build an AC, the classifier training kit is first trained and estimated. Then, the "Relative Weight" of the misclassified training data is increased. The second classifier is trained with these increased weights and re-estimated. Weights are updated again and continue as such. When all forecasters are trained, estimates are made by bagging or pasting methods, taking into account the accuracy ratios of all forecasters [6].

8) Gradient Boosting Classifier (GBC): It is a machine learning technique for gradient increasing, regression, and classification problems. A combination of weak prediction models creates a model typically made up of DT. The purpose of any supervised learning algorithm is to define and minimize a lost function.

9) Bagging Classifier: It is a method that aims to retrain the basic learner by deriving new education sets from an existing education set. Sampling is done by putting it in its place. The training set is produced by random selection by replacing the training set with n samples from the training set consisting of n samples in Bagging. Each selected sample is put back into the training set. Some examples are not included in the new training set, while others take place more than once. With the training sets chosen by replacing them randomly, successful elementary learners are trained to provide separation. Thus, collective success is achieved.

10) Multi-Layer Perceptron Classifier (MLP): It is a supervised learning approach and consists of an artificial neural network model. In this approach, input datasets are mapped to the appropriate set of outputs. It is entirely dependent on an MLP of a routed chart consisting of multiple layers of nodes and the next in each node. Each input node is called a neuron with a nonlinear activation function, which has a direct effect on the performance of the system [20]. The sigmoidal units of the hidden layer learn about the functions. For educational purposes, MLP uses a technique called backpropagation. In this study, the maximum number of iterations was selected as 1000.

D. Proposed Model

As described in the dataset section, the CM1, KC1, KC2, JM1, and PC1 datasets were used from the PROMISE warehouse [14]. The flow diagram applied to the model is shown in Fig. 3.

Data sets have passed through this flowchart, respectively. First, the data has passed through data processing and normalization steps. There are two main purposes of the normalization process. Eliminate data replays in the database and increase data consistency, normalization equation as in:

$$X_{new} = (X - X_{min})/(X_{max} - X_{min}) \quad (1)$$

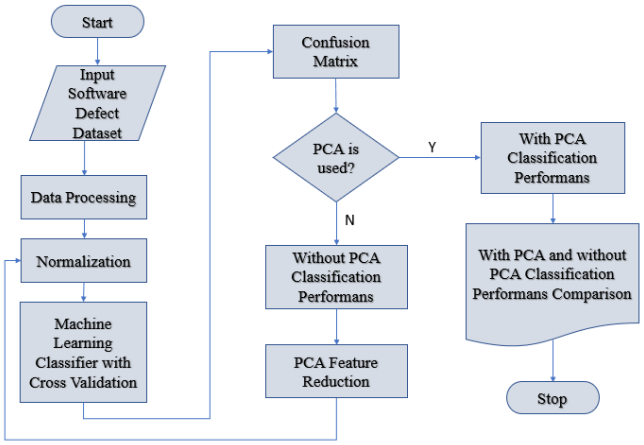

Fig. 3. The Flow Diagram Applied to the Model

Data, which becomes more consistent with normalization, is trained by the Cross-Fold Validation (CV) method. CV is a method used to assess the achievements of machine learning models. In this method, the data set is divided into training and test set, and the method selected for this process significantly affects the success of the model [11].

The reason for using CV in this study; In this way, it is trained in all conditions of the model, which is very necessary for the accuracy of the model. It helps to use the data better and gives more information about algorithm performance.

A k value is set before this method is used. In this study, k value is selected 5. The dataset is divided into 5 equal parts; each time the test data shifts one step to ensure that the tested data is also used within the test, as shown in Fig. 4. The result has been produced separately from each process. By obtaining the arithmetic average of these results, a general result of error or success metric has been obtained.

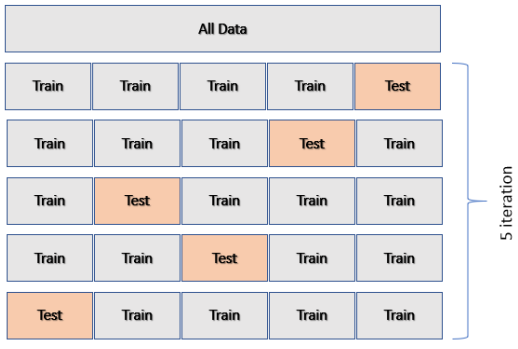

Fig. 4. Use to CV Model

The confusion matrix was used in the system at this stage. The CV process divided the data into five equal parts. And each piece is trained in order. The result was then reflected in the confusion matrix by comparing the data allocated for the test. The confusion matrix is the most commonly used method for evaluating the performance of models obtained from previously determined data sets of target data in machine learning [10]. The matrix is called the confusion matrix, which shows the extent to which I classify the test data set that contains positive and negative instances of the model, as shown in Fig. 5.

- True Positive (TP): The value in the test data is the same as the class the model values. The correct classification has been made.
- False Negative (FN): The value in the test data differs from the class produced by the model. The positive first is classified as negative. The incorrect classification was made.
- False Positive (FP): While the actual value is negative, it is classified as positive. The incorrect classification was made.
- True Negative (FN): While the actual value is negative, it is classified as negative. The correct classification has been made.

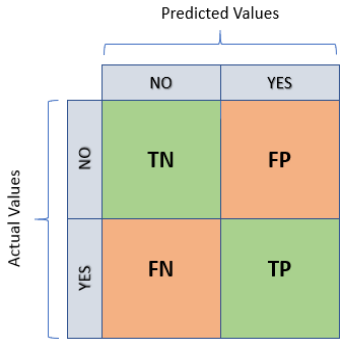

Fig. 5. Confusion Matrix Model

Along with the confusion matrix, the precision data, recall score, and accuracy values of the current data set were calculated.

$$Accuracy = (TP + TN) / (TP + TN + FP + FN) \quad (2)$$

$$Recall = TP / (TP + FN) \quad (3)$$

$$Precision = TP / (TP + FP) \quad (4)$$

After all these calculations are finished, the values obtained are written to the output, and the system asks the question: "Principal Component Analysis (PCA) is used?". If the answer is no, the system performs the same operations from the normalization step for all data sets. If the answer is yes, the system proceeds to the comparison step of the results.

PCA is a mathematical technique of explaining information in a multi-variable dataset with fewer variables and minimal loss of data. Another definition, [21] PCA is a conversion technique that allows the size of the dataset to be reduced to a smaller size by preserving the dataset in the dataset, which contains variables associated with a large number of interconnected. PCA reduces dimensionality in oversized datasets. The technique aims to reduce the number of variables in the dataset in the size reduction process. The variables obtained after the conversion are called the main components of the first variables. As the first base component, the variance value is selected with the largest, and other basic components are sorted to decrease variance values.

IV. EXPERIMENTS AND RESULTS

In this Section, all the results of the tests performed by the proposed system without using and using the feature reduction of all data sets are described and are shown in graphics. In this study, the number of features, which was 21, was reduced to 15 by using PCA. And better results have been observed. After completing the same processes within the system with PCA, the system outputs have started to compare.

The graph of the accuracy results of the 10 algorithms used in the system according to the PC1, CM1, KC1, KC2 and JM1 datasets, respectively, without using the PCA method, is shown in Fig. 6.

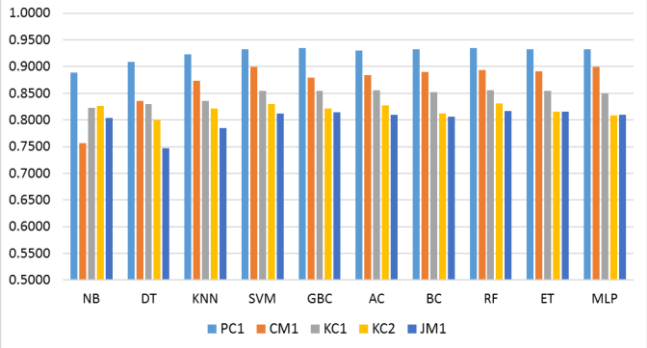

Fig. 6. Accuracy Result without PCA

The graph of the accuracy results of the ten algorithms used in the system, according to PC1, CM1, KC1, KC2, and JM1 datasets, respectively, using the PCA method is shown in Fig. 7.

Fig. 7. Accuracy Result with PCA

The accuracy results given in the algorithms used for the PC1 dataset with and without PCA are shown in Fig. 8.

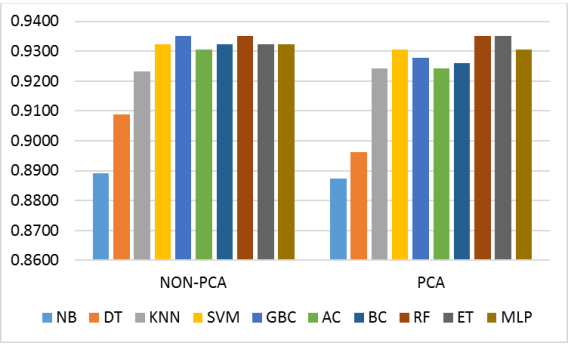

Fig. 8. Accuracy Result for PC1 dataset

The accuracy results given in the algorithms used for the CM1 dataset with and without PCA are shown in Fig. 9. The study mentioned in the Proposed Model section, where cv was used, and the k value was selected at 5. Accuracy, recall, precision, TP, TN, FP, FP values are calculated in each iteration when the work is complete. This calculation was done both using the PCA method and without the use of the PCA method.

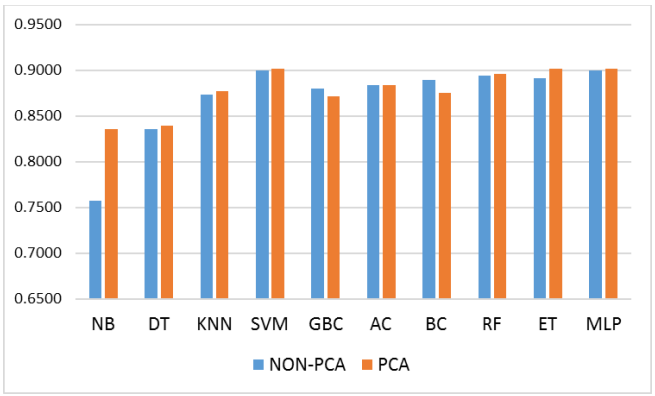

Fig. 9. Accuracy Result for CM1 dataset

The algorithms used in the study; Average values on datasets are shown in Table 3 without using the PCA method. The average values of datasets using the PCA method are shown in Table 4. For example, when calculating values; The accuracy value of the PC1 dataset is the arithmetic average of the sums of accuracy results in all algorithms.

TABLE III. PERFORMANCE METRIC WITH WITHOUT PCA (DATASET)

Data	Acc.	Rec.	Pre.	TP	TN	FP	FN
PC1	**0.925**	0.429	0.234	18	1008	59	24
CM1	0.871	0.241	0.143	7	427	42	22
KC1	0.846	0.506	0.258	84	1701	242	82
KC2	0.820	0.594	0.383	41	387	66	28
JM1	0.802	0.470	0.191	401	8324	1702	453

TABLE IV. PERFORMANCE METRIC WITH PCA(DATASET)

Data	Acc.	Rec.	Pre.	TP	TN	FP	FN
PC1	**0.922**	0.342	0.151	12	1011	65	21
CM1	0.878	0.246	0.114	6	432	43	17
KC1	0.850	0.531	0.267	87	1706	239	77
KC2	0.823	0.606	0.387	41	388	66	27
JM1	0.803	0.475	0.190	400	8335	1703	442

TABLE V. PERFORMANCE METRICS WITHOUT PCA (ALGORITHMS)

Data	Acc.	Rec.	Pre.	TP	TN	FP	FN
NB	0.811	0.436	0.237	126	2328	406	163
DT	0.775	0.360	0.360	192	2152	341	340
KNN	0.806	0.429	0.295	157	2282	375	209
SVM	0.830	0.632	0.086	46	2464	486	27
GBC	0.831	0.570	0.166	88	2425	444	67
AC	0.828	0.543	0.151	81	2424	452	68
BC	0.824	0.504	0.248	132	2361	400	130
RF	**0.833**	0.564	0.247	132	2390	401	102
ET	0.831	0.550	0.242	129	2386	403	105
MLP	0.827	0.671	0.034	18	2482	514	9

The algorithms used in the study and the average values by algorithms are shown in Table 5 without using the PCA method. On the other side, the effect of the PCA usage is shown in Table 6.

TABLE VI. AVERAGE VALUES OF BASED ON ALGORITHM WITH PCA

Data	Acc.	Rec.	Pre.	TP	TN	FP	FN
NB	0.810	0.431	0.240	128	2322	404	169
DT	0.779	0.372	0.373	199	2157	334	334
KNN	0.811	0.443	0.279	149	2304	384	187
SVM	0.830	0.602	0.111	59	2452	473	39
GBC	0.831	0.567	0.170	91	2422	442	69
AC	0.826	0.519	0.169	90	2408	442	84
BC	0.826	0.516	0.235	125	2374	407	117
RF	**0.837**	0.599	0.238	127	2406	406	85
ET	0.836	0.590	0.231	123	2406	409	86
MLP	0.823	0.000	0.000	0	2491	532	0

V. CONSLUSIONS

Software defect prediction is one of the important processes in Software Engineering. For creating an error-free or minimum error software, the correct prediction of errors in the software (specifically in its modules) has a vital importance. With the use of appropriate prediction models, software managers can increase the efficiency for their workers and allocate more time not only for testing but also for maintenance of the software systems.

In the literature, different approaches are used for this critical task. For the prediction of software defects, we prefer the use of 10 machine learning models as Decision Tree, Naive Bayes, K-Nearest Neighbor, Support Vector Machine, Random Forest, Extra Trees, Adaboost, Gradient Boosting, Bagging, and Multi-Layer Perceptron, on the public datasets CM1, KC1, KC2, JM1, and PC1 from the PROMISE warehouse. The comparative works showed that the used algorithms have better average accuracy rates for *PC1* dataset and *Random Forest* learning models with PCA approach gives better average performance for the used datasets. The reached performance values show that the proposed models have good accuracy for the prediction of software defects.

REFERENCES

[1] Z. Tian, J. Xiang, S. Zhenxiao, Z. Yi and Y. Yunqiang, “Software Defect Prediction based on Machine Learning Algorithms,” 5th International Conference on Computer and Communications (ICCC), Chengdu, China, pp. 520-525, December 2019.

[2] L. Y. Banowosari and B. A. Gifari, “System Analysis and Design Using Secure Software Development Life Cycle Based On ISO 31000 and STRIDE. Case Study Mutiara Ban Workshop,” 2019 Fourth International Conference on Informatics and Computing (ICIC), Semerang, Indonesia, October 2019.

[3] iso25000.com. [Online]. Available: https://iso25000.com/index. php/en/iso-25000-standards/iso-25010. [Accessed: 14-Apr-2020].

[4] F. Akba, A. Ucan, E. A. Sezer and H. Sever, “Assessment of Feature Selection Metrics for Sentiment Analyses: Turkish Movie Reviews,” 8th European Conference on Data Mining, Lisbon, Portugal, pp. 180-184, July 2014.

[5] L. Pelayo and S. Dick, “Applying Novel Resampling Strategies to Software Defect Prediction,” Fuzzy Information Processing Society NAFIPS ’07, San Diego, USA, pp. 24-27, June 2007.

[6] S. Aleem, L. F. Capretz and F. Ahmed, “Benchmarking Machine Learning Techniques for Software Defect Detection,” International Journal of Software Engineering & Applications, vol. 6, no. 3, pp. 11-23, 2015

[7] S. Wang and X. Yao, “Using Class Imbalance Learning for Software Defect Prediction,” IEEE Transactions on Reliability, vol. 62, no. 2, pp. 434-443, 2013.

[8] M. Prasad, L. Florence and A. Arya, “A Study on Software Metrics Based Software Defect Prediction using Data Mining and Machine Learning Techniques,” International Journal of Database Theory and Application, vol. 8, no. 3, pp. 179-190, 2015.

[9] D. Tomar and S. Agarwal, “Prediction of Defective Software Modules Using Class Imbalance Learning, ” Applied Computational Intelligence and Soft Computing, p.12, 2016.

[10] G. Batista, R. Prati and M. Monard, “A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data,” ACM SIGKDD Explor Newsl, vol. 6, no. 1, pp. 20-29, 2004.

[11] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, “Benchmarking Classification Models for Software Defect Prediction: A proposed Framework and Novel Findings,” IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 485-496, 2008.

[12] Y. Tekin and O. K. Sahingoz, "A Publish/Subscribe messaging system for wireless sensor networks," 2016 Sixth International Conference on Digital Information and Communication Technology and its Applications (DICTAP), Konya, 2016, pp. 171-176.

[13] D. R. Ibrahim, R. Ghnemat and A. Hudaib, “Software Defect Prediction using Feature Selection and Random Forest Algorithm,” International Conference on New Trends in Computing Sciences (ICTCS), Amman, Jordan, pp. 252-257, October 2017.

[14] Promise datasets page. [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasets-page.html. [Accessed: 21-Apr-2020].

[15] A. Koru and H. Liu “Building Effective Defect-Prediction Models in Practice,” IEEE Software, vol. 22, no. 6, pp. 23-29, 2005.

[16] C. Catal and B. Diri, “Investigating Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction Problem,” Information Sciences, vol. 179, no. 8, pp. 1040–1058, 2009.

[17] M. Anbu and G. S. Mala, “Feature Selection Using Firefly Algorithm in Software Defect Prediction,” Cluster Computing, vol. 22, pp. 10925–10934, 2017.

[18] X. Yang, K. Tang and X. Yao, “A learning-to-rank approach to software defect prediction,” Reliab. IEEE Trans, vol. 64, no. 1, pp. 234–246, 2015.

[19] A. K. Verma and S. Pal, “Prediction of Skin Disease With Three Different Feature Selection Techniques Using Stacking Ensemble Method,” Appl. Biochem. Biotechnol, pp 341-359, 2019.

[20] G. Karatas and O. K. Sahingoz, "Neural network based intrusion detection systems with different training functions," 2018 6th International Symposium on Digital Forensic and Security (ISDFS), Antalya, 2018, pp. 1-6.

[21] X. Jin and R. Bie, “Random Forest and PCA for Self-Organizing Maps Based Automatic Music Genre Discrimination,”. Conference on Data Mining, Las Vegas, Nevada, pp. 414–417, June 2006.

[22] G. Karatas, O. Demir and O. Koray Sahingoz, "Deep Learning in Intrusion Detection Systems," 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT), Ankara, Turkey, 2018, pp. 113-116.

[23] S. I. Baykal, D. Bulut and O. K. Sahingoz, "Comparing deep learning performance on BigData by using CPUs and GPUs," 2018 Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT), Istanbul, 2018, pp. 1-6.

[24] F. Akmel, E. Birihani and B. Siraj, “A Literature Review Study of Software Defect Prediction using Machine Learning Techniques,” International Journal of Emerging Research in Management & Technolgy, vol. 6, no. 6, pp. 300-306, 2017.