Dynamic Detection of Software Defects Using Supervised Learning Techniques

Article · April 2019

CITATIONS
0

READS
220

1 author:

Nahla Ibraheem Jabbar
University of Babylon
13 PUBLICATIONS **78** CITATIONS

SEE PROFILE

Dynamic Detection of Software Defects Using Supervised Learning Techniques

Alaa Al-Nusirat [1], Feras Hanandeh [2], Mohammad Kharabsheh[3], Mahmoud Al-Ayyoub[4] and Nahla Al-dhufairi[5]

[1, 2 and 3] Faculty of Prince Al-Hussein Bin Abdallah II For Information Technology, The Hashemite University, Jordan
[4] Faculty of Computer and Information Technology, Jordan University of Science and Technology, Jordan
[5] School of Engineering, Babylon University, Iraq

***Abstract*:** In software testing, automatic detection of faults and defects in software is both complex and important. There are different techniques utilized to predict future defects. Machine learning is one of the most significant techniques used to build such prediction models. In this paper, we conduct a systematic review of the supervised machine learning techniques (classifiers) that are used for software defect prediction and evaluate their performance on several benchmark datasets. We experiment with different parameter values for the classifiers and explore the usefulness of employing dimensionality reduction techniques, such as Principle Component Analysis (PCA), and Ensemble Learning techniques. The results show the effectiveness of the considered classifiers in detecting bugs. Additionally, using PCA did not have a noticeable impact on prediction systems performance while parameter tuning positively impact classifies' accuracy, especially with Artificial Neural Network (ANN). The best results are obtained by using Ensemble Learning methods such as Bagging (which achieves 95.1% accuracy with Mozilla dataset) and Voting (which achieves 93.79% accuracy with kc1 dataset).

***Keywords*:** Software defects prediction, Machine learning, Artificial Neural Network, supervised classifiers.

1. Introduction

Machine learning (ML) is a branch of Artificial Intelligence (AI) concerned with "teaching" computers how to act without being explicitly programmed for every possible scenario [2]. The main concept in ML is developing algorithms that can self-learn by training on a very large number of inputs (possibly with known results) [1] [3].

One of the major types of ML is supervised learning. This type relies fundamentally on estimating the future instances based on known instances. The aim of supervised learning is to extract a pattern of the distribution of class labels which rely on predictor features. This pattern is used to select class labels to the testing instances. Class labels for these instances are selected based on the predictor features that are known. Since features can be large in number and some of them can be less informative than the others, Feature Selection (FS), which is also called attribute selection, is a fundamental phase to build any prediction model.

Detecting bugs and fixing them are reflected positively on the software quality while locating the bugs increases the capacity of systems developing and increase the software reliability and efficiency [4]. Approaching the problem of bug detection as a supervised learning problem is a common trend in the literature.

According to many studies carried out in the field of software testing [6] [7], there many metrics of the source code that can be considered as features/attributes to identify defects. The features are extracted using several measures/techniques such as McCabe, Halstead, Line Count, Operator, and Branch Count, etc. Since there is a large number of such software metrics that can be used as features, FS is used to get the most relevant and informative ones in order to enhance the prediction accuracy [5].

Most of the information technology companies, such as Ericsson, Samsung and Microsoft, need to purpose high quality and error-free products. Since the number of defects in the products can be an index of the quality of the software, these companies have focused on the issues related to the defects in their products. Predicting software faults at an early phase of the software life cycle helps to reduce the costs of product development and maintenance.

There are two types of analysis can be carried out to predict software defect based on code metrics. The first is detecting whether code segment contains bugs or not. The second is estimating the potential impact of the defect taking into account various points of view such as density, severity or priority. In this paper, we focus primarily on the first type. However, it also includes some of the experiments involving the second type.

The objective of this research is to improve and develop an effective and accurate model for automatic software defect prediction. The contributions of this paper include presenting a comprehensive study of the software defect prediction algorithms, proposing a new model based on using different preprocessing and FS algorithms to obtain high effective and efficient model. Another significant contribution of this research is presenting an Ensemble Learning method for software defect prediction.

The rest of this paper structured as: In Section 2 some related work has been addressed. Then, the methodology is introduced in Section 3. We conclude our work in addition to the future works in the last section.

2. Related Work

Several methods have been introduced in the literature for the software defect prediction, and each has its own cons and pros according to speed, accuracy, and cost, etc. These methods included the work of [8], the authors carried out an analytical study which aims to compare and evaluate the performance of four types of the classifier which are Random Forest, Naïve Bayes, RPart and SVM. All the datasets were taken from the widely used NASA projects.

Selvaraj et al. in 2013 [9] used the Support Vector Machine (SVM) for predicting software fault. They compared the performance of SVM with Naïve Bayes model and Decision stumps. SVM performed better than Naïve Bayes model and Decision stumps, this study evaluated the performance of

SVM with different kernels based on six measures which are: accuracy, detection probability (dp) or recall, precision, Probability of false alarm (pf), and effort. KC1 dataset from NASA Metrics Data is used. It yielded recall = 0.86, accuracy = 86.05, F-Measure = 0.80, and precision = 0.804.

Another approach based on combining more than one machine learning technique has also shown in the literature [10]. Amardeep and Kaur paper 2014 attempted to evaluate the performance of the predictive accuracy models in the literature. The study proceeded by a principle of an area under the curve (AUC). This approach depends on analyzing prediction model that built using three common methods which are; Bagging, Boosting, and Rotation Forest. The experiments were conducted using15 Java codes from PROMISE database.

Several researchers adjust Neural Network learning and support vector machine algorithms to build bug prediction model [11]. They suggested a hybrid model using these two classifiers to increase the efficiency of bug prediction techniques. Their results were conducted using three different datasets from NASA to demonstrate that using PNN with SVM outperformed the use of other machine learning algorithms, for example, the accuracy of the proposed method with CM1 dataset was 90.16% which increases 5% over using other classification algorithms.

In 2000 [12] Zhang presented a comprehensive study about machine learning algorithms, and how each algorithm can actively contribute to the tasks of software engineering. In addition, he summarized the steps of the algorithm to perform each software engineering task. To enrich the topic, Zhang presented more detail in his book in 2003 [13].

In 2014 [14], Petkovic et. al. used a machine learning model to build an estimation and prediction criteria of the efficiency of distributed teamwork in a software engineering course. To realize their goal, they used the random forest classification (RF) model, by following a number of steps. Their results showed that RF is a good model in the prediction of software engineering process and performance.

The techniques used in their research in 2016 by Tsakiltsidis et al. [15] were similar to my own techniques, which concentrate on analyzing performance defects early, the authors used the mobile advertisement and marketing domain to find defect prediction model which is handy and ideal. Therefore, they utilized four classifiers to build several models; which are; C4.5 Decision Trees, Naive Bayes, Bayesian Networks, and Logistic Regression. Outcomes proved that a C4.5 model is the best model that can be used to predict performance defects, it yielded recall = 0.73, accuracy = 0.85, and precision = 0.96, such that lines of code changed (lines of code added or removed), file age and file size were used as explanatory variables.

In this study [16], the authors focused on code clones which can be defined as a code segment that frequently more than one time in the same program that resulted in a longer program with high complexity time, so their thought was to use support vector machine in order to estimate the identity between two code-clones. The authors applied their experiments on C source codes as inputs to extract features from them, after that the authors used Tkinter tool for classification.

The implementation of software metrics and software visualization after combination helps in a reverse engineering field. Michele Lanza (2001) [17] approach proposed an effective way to recover the development of a matrix of software from version to another. This matrix can minimize the quantity of the needed data in analyzing software development. This estimation matrix is used fundamentally to imagine the evaluation of categories in object-oriented systems. But the proposed method has some limitations such as negative effects on the number of changes between the two versions, missing of scalability in generating large numbers of classes, and changing the class name which will lead to deal with it as a new different approach.

In this study [18], the authors have done a performance analysis on some classification algorithms including Bayesian Logistic Regression, Hidden Naïve Bayes, Radial Basis Function (RBF)Network, Voted Perceptron, Lazy Bayesian Rule, Logit Boost, Rotation Forest, NNge, Logistic Model Tree, REP Tree, Naïve Bayes, Multilayer Perceptron, Random Tree and J48. The rendering of the algorithms was deliberate in terms of Accuracy, Precision, Recall, FMeasure, Root Mean Squared Error, Receiver Operator characteristics Area and Root Relative Squared Error using WEKA data mining tool. To hold a balanced judgment of the classification algorithms' performance, no feature selection or performance boosting method was appointed. The research display that a number of classification algorithms exist that if duly examined through feature selection means will produce more accurate results for email classification [23]. Rotation Forest has shown a near degree to achieve the most accurate result. Rotation Forest is set to be the classifier that accords the best accuracy of 94.2%. Though none of the algorithms didn't achieve 100% accuracy in sorting spam emails, Rotation Forest has shown a near stage to obtain the most accurate result.

3. Methodology

This section, characterize the researcher's procedure which is used for the purposes of software defect prediction that is theorized as one of the most important issues related to the quality of software.

This work suggests several models depending on supervised learning methods. In this paper, the proposed work aims to build a set of models that can correctly predict the class of the different source code metrics.

The input to these models is a set of source code metrics which is called training data, these metrics belong to defect or non-defect classes and a set of attributes that describing different characteristics of the source code (i.e., line count of code, difficulty, design complexity, etc.). Those attributes can be used to estimate the unlabeled source code for which class information is belongs to them.

The methodology of this paper will be presented in the next subsections. These subsections include the

1. Feature Selection Technique: is an essential phase to build any prediction model. The goal of this step to choose the minimum number of input features and determine the features that are the most efficient in improving the performance of prediction. The features were extracted using various techniques such as McCabe, Halstead, Line Count, Operator, and Branch Count…etc.

2. Preprocessing: Principal Component Analysis (PCA) [19] is used to solve the dimensionality reduction problem, it is a preprocessing step for data before the use of supervised

machine learning classifiers. This operation helps in reducing the time complexity and increasing efficiency of classifiers.

3. The Ensemble Learning: The proposed approach is based on combining two different classifiers namely: (Decision Tree) (Support Vector Machine) to get a more accurate result than a single classifier.

Figure 1 summarizes the steps of our proposed methodology.

In **Figure 1**, shows the major steps of defect prediction model which is mainly divided into many stages the author reads the report of defects from open source code, extracts features stage that includes basic complexity, independence of the program, the complexity of the design and lines of code.

Classify training model for extracted features. Apply supervised classifiers to predict bugs. The last stage is the decision whether the class is a defect or non-defect.

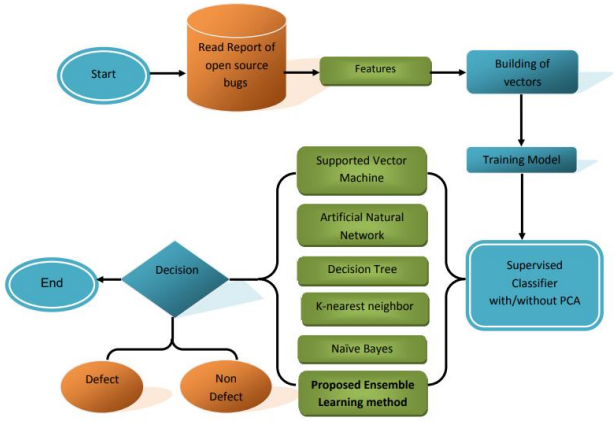

Figure 1. Major steps of defect prediction model

This work investigates the effect of implementing various classifiers on the performance of the software defect prediction system. The feature selection for representation of source code is built is based on standard techniques such as McCabe and Halstead.

According to many studies carried out in the field of software metrics [20] [21], the metric of source code was considered as features to identify defects. The features were extracted using various techniques such as McCabe, Halstead, Line Count, Operator, and Branch Count…etc.

4. Experimental Results

In this paper conducted four experiments over deferent datasets that were taken from PROMISE and NASA databases repository datasets which are described below:

NASA datasets are publically available in the NASA repository by NASA Metrics Data Programme. According to Malhotra's study in 2015 [22], this dataset is used in 60% of the selected primary studies.

We used five standard datasets; mozilla4 dataset, ar1, kc1, ar4 and pc1 dataset.

- **Mozilla4**: this dataset is written in C++ language, It consists of 15544 modules and 5 static code attributes (McCabe, Halstead and LOC measures) and one defect information (0/1).
- **Kc1**: this dataset is written in C++ language, it has included 43 KLOC, number of modules is 2108 and defective modules 325
- **Pc1** : this dataset is written in C language, it has included 40 KLOC, number of modules is 1107 and defective modules 76
- **Ar1 & Ar4**: These datasets are written in C language, each of them consists of 121 modules (9 defective / 112 defect-free) and 29 static code attributes, and one defect information (false for defect /true for non-defect). In **Table 1** below summarizes the specifications of each used dataset.

Table 1. The datasets used in this study

Dataset	# of modules	Language	# of attributes
mozilla4	15544	C++	6
Kc1	2108	C++	43
Ar1	121	C	30
Ar4	121	C	30
Pc1	1107	C	40

One of the most popular public domain data mining tools is the Waikato Environment for Knowledge Analysis (WEKA) tool is open source Java code, this tool is widely used for classification purposes to implement the supervised machine learning techniques. In this paper, used WEKA 3.7 version to conduct empirical results for evaluating the efficiency of five supervised classification techniques in software bug prediction. This tool is available online freely. WEKA supports a well-tested and reliable Java implementation of many methods for all type of tasks required by most machine learning and data mining issues.

Our experiments are divided into four main parts: first part is the results of implementation five well-known supervised classifiers with the default settings for all algorithms. The second part is the implementation of adaptive supervised classifiers, where some of the parameters are modified; in the third part, the researcher presented comparisons of the supervised classifiers after using PCA preprocessing methods. Finally, the researcher presented our proposed Ensemble Learning techniques that are combined two supervised classifiers. The researcher tested all algorithms by using the 10-fold cross-validation method. The results are analyzed in terms of accuracy, precision, recall and F1-measure.

Experiment 1: Results of Supervised Classifiers with Default Settings

In this experiment, we used five standard datasets to display the results, knn achieved good results with ar1, kc1, pc1 datasets, and the accuracy obtained was (90.0826%), (95.1724%) and (92.0649%) respectively. As we see, the F-measure results are equal 0.901, 0.946 and 0.921 respectively, whereas the accuracy was (79.4393%) with ar4 dataset, and **Table 2** shows the gained results.

Table 2. Results of Implementing KNN using default settings

Dataset	Accuracy	Recall	Precision	F-measure
Mozilla4	88.9932%	.890	.890	.890
ar1	90.0826%	.901	.901	.901
kc1	**95.1724%**	**.952**	**.944**	**.946**
ar4	79.4393%	.794	.771	.780
pc1	92.0649%	.921	.922	.921

A sample of the results is shown in **Table 3** below. As shown, mozilla4, ar1, kc1, pc1 datasets resulted respectively

in 94.5642%, 92.562%, 94.4828% and 93.5978% accuracy with environment prediction.

Table 3. Results of Implementing Decision Tree using default settings

Dataset	Accuracy	Recall	Precision	F-measure
mozilla4	**94.5642%**	**.946**	**.947**	**.945**
ar1	92.562%	.926	.857	.890
kc1	94.4828%	.945	.893	.918
ar4	81.3084%	.813	.806	.809
pc1	93.5978%	.936	.922	.924

Table 4 shows the gained results. The results below show that all datasets achieved good results except pc1 with accuracy (74.4%), such that the accuracy of the rest datasets was in the range [81 - 93] %, and the f-measure was in the range [80 - 93] %.

Table 4. Results of Implementing BayesNet using default settings

Dataset	Accuracy	Recall	Precision	F-measure
mozilla4	**93.522%**	**0.935**	**0.935**	**0.935**
ar1	90.9091%	0.909	0.856	0.882
kc1	87.5862%	0.876	0.927	0.897
ar4	81.3084%	0.813	0.806	0.809
pc1	74.3913%	.744	.900	.803

After analyzing **Table 5**, the results show that kc1 got 93.7931% accuracy. The average of four measures obtained against five bug prediction datasets indicates that the SVM algorithm achieved stable results. However, and for Mozilla4 dataset, the SVM classifier has 10%, 10%, 9% and 10% lower Accuracy, Recall, Precision and F1-measure than NB respectively.

Table 5. Results of Implementing SVM using default settings

Dataset	Accuracy	Recall	Precision	F-measure
Mozilla4	83.21%	.832	.848	.836
ar1	91.7355%	.917	.856	.886
kc1	**93.7931%**	**.938**	**.917**	**.924**
ar4	85.0467%	.850	.838	.823
pc1	92.9666%	.930	.866	.897

The Table 6 below shows that all datasets achieved good results except for ar4 with accuracy (81.3%) such that the accuracy of the rest datasets was in the range [90 – 93.5] %, and the f-measure was in the range [89 - 93] %, another notable result which was reported is that all measures vary among datasets.

Table 6. Results of Implementing ANN using default settings

Dataset	Accuracy	Recall	Precision	F-measure
mozilla4	91.1869%	.912	.911	.911
ar1	90.0826%	.901	.890	.895
kc1	93.1034%	.931	.931	.931
ar4	81.3084%	.813	.799	.805
pc1	**93.5978%**	**.936**	.921	.917

For example, "pc1" dataset has a prediction F1-measure of 91.7% while the same dataset has a noticeably poor Recall of 80.1% using BayesNet. These poor results indicate that the "pc1" dataset is highly overlapped with other datasets. Finally, ANN and NB classifiers perform highly in the **ar1** dataset.

Experiment 2: Results of Adaptive Supervised Classifiers with Adjusted Settings

After analyzing **Table 7**, the researcher found that using Manhattan distance little bit greater than Euclidian distance with regards to F1-measure in ar4 and mozilla4.

Table 7. Results of Implementing KNN using modified settings

Dataset	Distance	Accuracy	Recall	Precision	F-measure
Mozilla 4	Euclidian	89.13%	0.891	0.892	0.892
	Manhattan	89.7523%	.898	.899	.898
ar1	Euclidian	90.08%	0.901	0.855	0.877
	Manhattan	89.2562%	.893	.854	.873
kc1	Euclidian	**95.17%**	**0.952**	**0.954**	**0.934**
	Manhattan	94.4828%	.945	.926	.929
ar4	Euclidian	84.11%	0.841	0.822	0.816
	Manhattan	85.9813%	.860	.851	.837
pc1	Euclidian	92.25%	0.922	0.908	0.914
	Manhattan	92.2453%	.922	.905	.911

The experimental results had shown in **Table 8** that mozilla4 dataset yielded 94.6092% accuracy, followed by kc1 with a closed result equals to 94.4828%.

Table 8. Results of Implementing Decision Tree using modified settings

Dataset	Accuracy	Recall	Precision	F-measure
Mozilla4	**94.6092%**	**.946**	**.947**	**.945**
Ar1	91.7355%	.917	.856	.886
Kc1	94.4828%	.945	.893	.918
Ar4	81.3084%	.813	.775	.779
Pc1	93.2372%	.932	.912	.909

This experiment is conducted using BayesNet, shows that the used search algorithm is LAGD Hill Climber- L 2-G 5- P 1 -S BAYES, **Table 9** shows the gained results. All datasets achieved good results except pc1 with accuracy (76.7358%), such as that the accuracy of the rest datasets was in the range [81.3 – 94.4] %, and the f-measure was in the range [80.9 – 94.3] %.

Table 9. Results of Implementing BayesNet using modified settings

Dataset	Accuracy	Recall	Precision	F-measure
mozilla4	**94.3969%**	**.944**	**.947**	**.943**
ar1	90.9091%	.909	.856	.882
kc1	87.5862%	.876	.927	.897
ar4	81.3084%	.813	.806	.809
Pc1	76.7358%	.767	.897	.819

This part of the second experiment tested SVM classifier with the adaptive settings, it's worthily mentioned that the filterType = Standardize training data with a number of kernel = 250007-G 11.0. **Table 10** shows the gained results after analyzing it the researcher found that kc1 dataset outperformed the other datasets with regards to accuracy and F1-measures which are equal to (**94.4828%**) and (**.918**) respectively.

The last phase of experiment two tested Neural Network classifier with the adaptive settings that are used to conduct it, such that validationSetSize = 43, and the randomSeed = 5. **Table 11** shows the gained results. The results below showed

that all datasets achieved good results except ar4 with accuracy (86.9159%), such as the accuracy of the rest datasets was in the range [90.4 – 95.86]%, and the f-measure was in the range [89 - 93]%.

Table 10. Results of Implementing SVM using modified settings

Dataset	Accuracy	Recall	Precision	F-measure
mozilla4	89.1991%	.892	.892	.892
ar1	90.9091%	.909	.856	.882
kc1	**94.4828%**	**.945**	**.893**	**.918**
ar4	81.3084%	.813	.661	.729
Pc1	93.688%	.937	.924	.918

Table 11. Results of Implementing ANN using modified settings

Dataset	Accuracy	Recall	Precision	F-measure
Mozilla4	90.4214%	.904	.904	.903
ar1	92.562%	.926	.857	.890
kc1	**95.8621%**	**.959**	**.960**	**.947**
ar4	86.9159%	.869	.870	.845
Pc1	93.147%	.931	.908	.907

The **Figures 1-6** below compare between default and modified settings for each dataset.

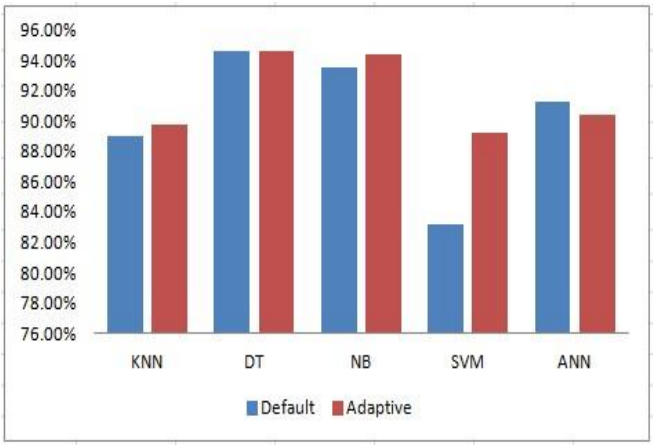

Figure 2. A comparison between default and adaptive classifiers using mozilla4 dataset

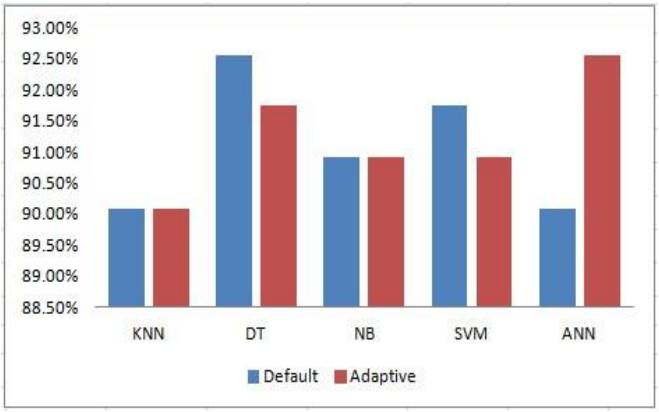

Figure 3. A comparison between default and adaptive classifiers using ar1 dataset

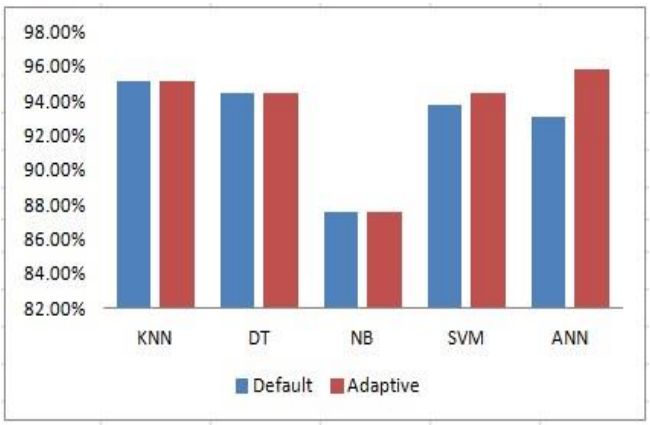

Figure 4. A comparison between default and adaptive classifiers using kc1 dataset

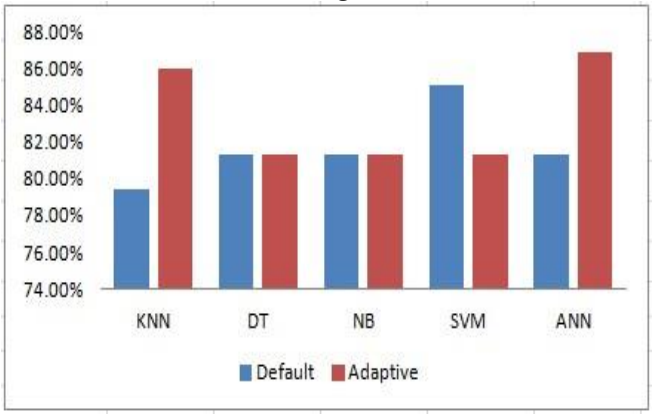

Figure 5. A comparison between default and adaptive classifiers using ar4 dataset

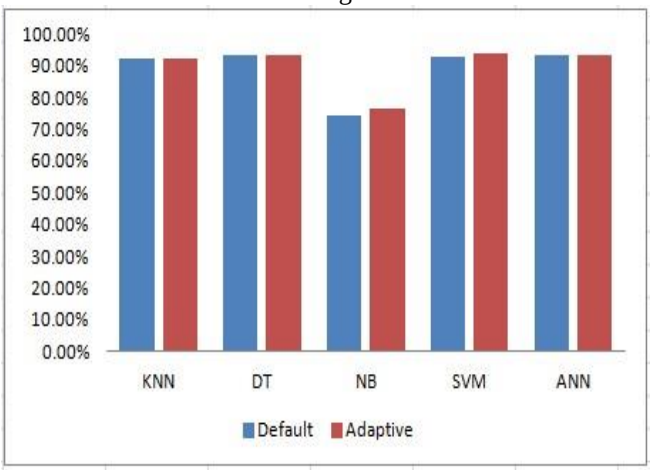

Figure 6. A comparison between default and adaptive classifiers using pc1 dataset

After analyzing **Figures 1-6** above, the researcher found that the adaptive classifier outperformed default settings on most datasets with regards to accuracy results. Recall results obtained showed that the **ar4** was positively affected by the modified settings of the classifier on two models which are, KNN and ANN.

Moreover, for the **ar1** dataset, the ANN classifier has (2.48%) higher accuracy than default settings.

Experiment 3: Results of Supervised Classifiers with Preprocessing Feature selection: PCA.

Table 12. Results of Implementing PCA with different classifiers with preprocessing Feature selection: PCA.

Dataset	Name of Classifier	Accuracy	Recall	Precision	f-measure
mozilla4	BayesNet	75.4969%	.755	.746	.746
	SVM	75.6835%	.757	.748	.746
	IBK	**85.4037%**	.854	.853	.853
	REPTREE	84.072%	.841	.839	.840
	Multilayer perceptron	81.0679%	.811	.807	.807
ar1	BayesNet	**92.562%**	.926	.857	.890
	SVM	92.562%	.926	.857	.890
	IBK	88.4298%	.884	.854	.869
	REPTREE	92.562%	.926	.857	.890
	Multilayer perceptron	89.2562%	.893	.887	.890
kc1	BayesNet	93.7931%	.938	.917	.924
	SVM	**95.1724%**	.952	.954	.934
	IBK	92.4138%	.924	.909	.916
	REPTREE	94.4828%	.945	.893	.918
	Multilayer perceptron	92.4138%	.924	.920	.922
ar4	BayesNet	85.0467%	.850	.835	.835
	SVM	82.243%	.822	.854	.751
	IBK	**86.9159%**	.869	.861	.863
	REPTREE	84.1121%	.841	.835	.800
	Multilayer perceptron	85.9813%	.860	.847	.848
pc1	BayesNet	90.1713%	.902	.894	.897
	SVM	93.0568%	.931	.866	.897
	IBK	92.2453%	.922	.919	.920
	REPTREE	92.4256%	.924	.892	.902
	Multilayer perceptron	**93.5077%**	.935	.919	.917

Table 12 shows the gained results that for the mozilla4 dataset, the accuracy achieved between 0.75 to 0.85%, were the best result gained with kNN with accuracy (85.4037%). Also, the same results were obtained with using ar1, between (88%) with KNN and (92%) with BayesNet.

Experiment 4: Results of Ensemble Learning Classifiers

In this experiment, a combination of classifiers with some major modifications has been applied. The researcher combined Bagging with REPTree classifier the researcher chose bagging for its efficiency and REPTree classifier for its flexibility and appropriateness to the real world scenarios. The aim of generating additional datasets in training phase is done by distributing a specified percentage of features in the training dataset with the remaining features in the test dataset; this approach is called Bagging.

Table 13. Results of Implementing Ensemble Learning methods

Dataset	Hybrid machine learning methods	Accuracy	Recall	Precision	F-measure
mozilla4	Bagging + REPTree	**%95.01**	0.95	0.952	0.949
ar1		%92.56	0.926	0.857	0.89
kc1		%93.79	0.938	0.892	0.915
ar4		%85.05	0.85	0.838	0.823
pc1		%93.51	0.935	0.919	0.919
mozilla4	Vote + REPTree + Ibk	%93.46	0.935	0.934	0.934
ar1		%92.56	0.926	0.857	0.89
kc1		**%93.79**	0.938	0.917	0.924
ar4		%84.11	0.841	0.822	0.822
pc1		%93.51	0.935	0.919	0.918

After analyzing **Table 13** above, it was noted that using those two Ensemble Learning methods achieved the best results compared to the previous three experiments. The highest accuracy is gained when applied over mozilla4 dataset. It was found that the Ensemble Learning classifier gained good results in the first three experiments with accuracy (95.1%) with an increase of (1%) accuracy over adaptive DT and Adaptive NB on most datasets with regards to accuracy results.

Finally, according to pc1, all the accuracy results were approximately (93%) except for (default and adaptive NB) with 74.39% and 76.73% respectively. The **Figure 7** below summarizes the comparison between default and Ensemble Learning model in terms of accuracy.

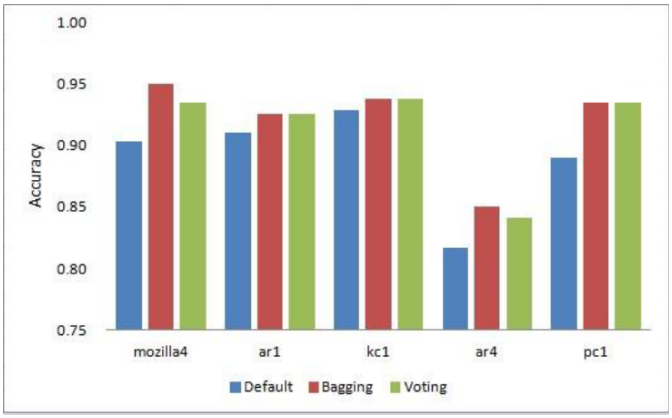

Figure 7. Comparison between default and Ensemble Learning model in terms of accuracy

5. Conclusions

This research deals with the task of software defect prediction. In this task, are given some of the predefined datasets that are organized in a defect/ non-defect structure. The target of defect prediction is to efficiently and effectively combine further information on classes' structure into the learning process.

The research is given in this paper concentrates on two aspects of bug prediction: learning and performance evaluation. We argue that software should be consistent with no bugs. Then, five learning algorithms are implemented carried out consistent classification. The results showed that the researchers use a decision tree for bug prediction studies and avoid using Naïve Bayes.

The main contribution of this research is the new Ensemble Learning software defect prediction model. For instance, when default Machine Learning classifier is applied on the mozilla4 dataset, the average accuracy was 90%, while 95% reached when applying the Ensemble learning method.

6. Future Work

Future studies will utilize feature selection method to resolve the problem of extracted a fully large number of metrics used as features and with hardness in identifying any of these features is the best way to realize the best performance of the predicting defect systems.

In a future study, research can be taken out to get better the quality of the software defect prediction model by applying other existing machine learning algorithms.

Recommendation for future study is to increase the size of the software defect prediction datasets since a larger pattern size will produce more accurate results.

7. Acknowledgement

The current study was submitted in partial fulfillment of the requirements for the degree of master for the lead author.

References

[1] M. Bkassiny, Y. Li ,SK. Jayaweera, " A survey on machine-learning techniques in cognitive radios," IEEE Communications Surveys & Tutorials, Vol. 15, No. 3, pp. 1136-59, 2013.

[2] F. Thung, S. Wang, D. Lo and L. Jiang, "An Empirical Study of Bugs in Machine Learning Systems," IEEE 23rd

International Symposium on Software Reliability Engineering, Dallas, pp. 271-280, 2012.

[3] J. S. Di Stefano and T. Menzies, "Machine learning for software engineering: case studies in software reuse, ", 14th IEEE International Conference on Tools with Artificial Intelligence, pp. 246-251, 2002.

[4] K. A. Gunes, and L. Hongfang, " Building effective defect-prediction models in practice," IEEE Software, Vol. 22, No. 6, pp. 23-29 , 2005.

[5] J. Śliwerski, T. Zimmermann, A. Zeller, " When do changes induce fixes? ," InACM sigsoft software engineering notes , Vol. 30, No. 4, pp. 1-5 , 2005.

[6] S. Agarwal and T. Divya, "A feature selection based model for software defect prediction. ," International Journal of Advanced Science and Technology, Vol. 65, pp. 39-58, 2014.

[7] V. Gupta, N. Ganeshan and T. K. Singhal, "Developing Software Bug Prediction Models Using Various Software Metrics as the Bug Indicators," International Journal of Advanced Computer Science and Applications, Vol. 6, No. 2, pp. 60-65, 2015.

[8] D. Bowes, H. Tracy and P, Jean,"Software defect prediction: do different classifiers find the same defects? ," Software Quality Journal, Vol. 26, No. 2, pp. 1-28, 2017.

[9] P.A .Selvaraj ,Dr.P. Thangaraj, "Support Vector Machine for Software Defect Prediction," International Journal of Engineering & Technology Research, Vol. 1, No. 2, pp. 68-76, 2013.

[10] A. Kaur and K. Kaur, "Performance analysis of ensemble learning for predicting defects in open source software," In Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on, IEEE, PP. 219-225, 2014.

[11] M. M .Askari and K. B. Vahid,"Software defect prediction using a high performance neural network," International Journal of Software Engineering and Its Applications , pp. 177-188, 2014.

[12] DU. Zhang, "Applying machine learning algorithms in software development,"Proceedings of the 2000 Monterey workshop on modeling software system structures in a fastly moving scenario, Vol. 26, No. 2, pp. 275-291, 2000.

[13] DU. Zhang and JP. Jeffrey , "Machine learning and software engineering. ," Software Quality Journal , Vol. 11, No. 2, pp. 87-119, 2003.

[14] D.Petkovic, M. Sosnick-Pérez, S. Huang, R.Todtenhoefer,K. Okada,S. Arora ,R. Sreenivasen,L. Flores, S. Setap Dubey, " Software engineering teamwork assessment and prediction using machine learning," In2014 IEEE Frontiers in Education Conference (FIE) , pp. 1-8, 2014.

[15] S.Tsakiltsidis, A. Miranskyy and E. Mazzawi, "On Automatic Detection of Performance Bugs," In Software Reliability Engineering Workshops (ISSREW), PP.132-139, 2016.

[16] S. Jadon, "Code clones detection using machine learning technique: Support vector machine," In Computing, Communication and Automation (ICCCA), PP. 399-303,2016 .

[17] M. Lanza, "The evolution matrix: Recovering software evolution using software visualization techniques," Proceedings of the 4th international workshop on principles of software evolution, PP 37-42, 2001.

[18] M.Shuaib,O. Osho, I.Ismaila,JK, "Alhassan , Comparative Analysis of Classification Algorithms for Email Spam Detection," International Journal of Computer Network and Information Security,VOL.10,NO.1,2018.

[19] S. Karamizadeh, S. Abdullah, S. M. Manaf, M.Zamani, and H. Alireza, "An overview of principal component analysis," Journal of Signal and Information Processing ,vol.4,no. 3, pp.173 , 2013.

[20] R.Moser, W. Pedrycz and G. Succi, " A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction,"In Proceedings of the 30th international conference on Software engineering ACM,PP. 181-190 , 2008.

[21] K. GaoT. M. Khoshgoftaar, H. Wang and N. Seliya, " Choosing software metrics for defect prediction: an investigation on feature selection techniques," Software: Practice and Experience, vol.41,no.5,pp. 579-606 ,2011.

[22] R. Malhotra, " A systematic review of machine learning techniques for software fault prediction," Applied Soft Computing, pp.504-518 , 2015.

[23] MS.Gadelrab, M.ElSheikh, MA. Ghoneim, M. BotCap. Rashwan, " Machine Learning Approach for Botnet Detection Based on Statistical Features," International Journal of Communication Networks and Information Security,vol.10,no.3,pp.563, 2018.