IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

COMPARATIVE EVALUATION OF MACHINE LEARNING METHODS' PERFORMANCE IN DETECTING SOFTWARE BUGS

[1]Aiswarya G, [2]Dr Sudheer S Marar
[1]MCA Scholar, [2]Professor & HOD
[1]Department of MCA
[1]Nehru College of Engineering and Research Centre,Pampady,India

Abstract : Machine knowledge styles may be used to dissect data from several shoes, allowing generators to recover applicable information [1]. Machine knowledge styles have been designed to be salutary in precluding software excrescencies [2]. In this discussion, we probe the relative efficacity of multiple machine knowledge algorithms for software disfigurement identification using extensively available datasets.The results showed that the maturity of machine knowledge types performed well on software bug datasets [3].

IndexTerms **– Machine Learning Methods, Predictive Analytics, Software Bug Detection.**

I. INTRODUCTION

As software technology advances, the quantity of software products grows, making maintenance a difficult chore. Maintenance activities account for more than half of a software system's lifecycle cost [4].As software systems become more complicated, the likelihood of having malfunctioning modules increases.It is critical to predict and rectify errors before they are provided to consumers because software quality assurance is a time-consuming operation that may not allow for thorough testing of the entire system owing to financial constraints. As a result, identifying a problematic software module can help us use our limited time and resources more effectively.

A bug demonstrates that the system behaves unexpectedly for a particular set of criteria. During software testing, the unexpected behavior is recognized and documented as a bug. A software bug is defined as an "imperfection in the software development process that would cause software to fail to meet the desired expectation"[6]. Furthermore, discovering and resolving errors is an expensive part of software development [7]. It has been found that the bulk of software defects are concentrated in a limited number of modules [8,9].Thus,timely identification of software vulnerabilities facilitates the efficient allocation of testing resources and allows developers to improve the architectural design of a system by identifying the high-risk regions of the system.

Machine learning approaches that can be used to find flaws in software datasets include classification and clustering. Classification is a data mining and machine learning approach that can help predict software bugs [10]. It entails categorizing software modules as imperfect or non-defective using a set of software complexity criteria and a classification model generated from previous development project data [11]. This research examines various machine learning algorithms for detecting software bugs and compares their performance [12]. The remainder of this paper is organized as follows: Section II discusses related work on the chosen research topic; Section III discusses the various machine learning techniques, data pre-processing and prediction accuracy indicators, experiment procedure, and results; Section VI discusses comparative analysis of different methods; and Section V concludes the study.

II. LITERATURE SURVEY.

Lessmann et al. [13] established a novel framework for software defect prediction by comparing classification algorithms on various datasets and found that their chosen classification approaches give high prediction accuracy and allow metrics-based classification.The results of the studies revealed that there is no significant variation in performance between different categorization algorithms. The study did not examine all machine learning techniques for software bug prediction. Sharma and Jain [14] investigated the WEKA technique for several classification algorithms but did not test it for software bug prediction. Kaur and Pallavi [15] investigated many data mining approaches for software bug prediction but did not give a comparative performance study of them.

RESEARCH METHODOLOGY

This study conducts a comparative performance analysis of several machine learning approaches for software bug prediction using publicly accessible data. Machine learning approaches have shown to be beneficial for predicting software bugs. Machine learning approaches are divided into two major groups to compare their performance, such as supervised learning vs unsupervised learning.

3.1 Datasets & Preprocessing

The investigations used datasets from the PROMISE data repository [16].Table 1 shows dataset details.NASA obtained the datasets from genuine software projects, which comprise a range of software components.We used public domain datasets in our experiments because they serve as a baseline for defect prediction research, allowing other researchers to easily compare their methodologies [17].Datasets were generated using a variety of programming languages and code metrics, including Halstead's complexity, code size, and McCabe's cyclomatic complexity, among others. A baseline experiment was performed. Experiments were done with the Waikato Environment for Knowledge Analysis (WEKA) tool.

3.2 Performance Measures

Performance measures such as accuracy, mean absolute error, and the F-measure based precision and recall were employed in the comparison study [18].Accuracy is defined as the total number of correctly recognized bugs divided by the total number of bugs,and it is determined using the formulae below:

Accuracy = (TP + TN) / (Total Number of Instances)

Accuracy (%) = (rightly classified software bugs/Total software bugs) * 100

Precision = TP / (TP + FP)

Table 1. Datasets Information

Dataset	Language Used	Code Size (LOC)	Number of Modules	Defect Count
CM1	C	20,000	**505**	48
JM1	C	315,000	10,878	2,102
KC1	C++	43,000	2,107	325
KC2	C++	18,000	522	105
KC3	Java	18,000	458	43
MC1	C++	63,000	9,466	68
MC2	C	6,000	161	52
MW1	C	8,000	403	31
PC1	C	40,000	1,107	76
PC2	C	26,000	5,589	23
PC3	C	40,000	1,563	160
PC4	C	36,000	1,458	178
PC5	C++	164,000	17,186	516
AR1	C	29,000	121	9
AR6	C	29,000	101	15

Table 2. Performance of several machine learning algorithms using cross-validation test mode grounded on accuracy

Datasets	Naive Bayes	Multi-Layer Perceptron (MLP)	Support Vector Machine(SVM)	Ada Boost	Bagging	Decision Trees	Random Forest Algorithm	J48	KNN	RBF	K-means
AR1	83.45	89.55	91.97	90.24	92.23	89.32	90.56	90.15	65.92	90.33	90.02
AR6	84.25	84.53	86.00	82.70	85.18	82.88	85.39	83.21	75.13	85.38	83.65
CM1	84.90	89.12	90.52	90.33	89.96	89.22	89.40	88.71	84.24	89.70	86.58
JM1	81.43	89.97	81.73	81.70	82.17	81.78	82.09	80.19	66.89	81.61	77.37
KC1	82.10	85.51	84.47	84.34	85.39	84.88	85.39	84.13	82.06	84.99	84.03
KC2	84.78	83.64	82.30	81.46	83.06	82.65	82.56	81.29	79.03	83.63	80.99
KC3	86.17	90.04	90.80	90.06	89.91	90.83	89.65	89.74	60.59	89.87	87.91
MC1	94.57	99.40	99.26	99.27	99.42	99.27	99.48	99.37	68.58	99.27	99.48
MC2	72.53	67.97	72.00	69.46	71.54	67.21	70.50	69.75	64.49	69.51	69.00
MW1	83.63	91.09	92.19	91.27	92.06	90.97	91.29	91.42	81.77	91.99	87.90
PC1	88.07	93.09	93.09	93.14	93.79	93.36	93.54	93.53	88.22	93.13	92.07
PC2	96.96	99.52	99.59	99.58	99.58	99.58	99.55	99.57	75.25	99.58	99.21
PC3	46.87	87.55	89.83	89.70	89.38	89.60	89.55	88.14	64.07	89.76	87.22
PC4	85.51	89.11	88.45	88.86	89.53	88.53	89.69	88.36	56.88	87.27	86.72

PC5	96.93	97.03	97.23	96.84	97.59	97.01	97.58	97.40	66.77	97.15	97.33
Mean	83.47	89.14	89.29	88.59	89.386	88.47	89.08	88.33	71.99	88.87	87.29

The equation for calculating recall is TP / (TP + FN).The F-measure is a combined measure of recall and precision, and the higher the value, the better the machine learning method's prediction accuracy.

F = (2 * precision * recall) / (Precision + recall)

3.3 Experiment Procedure and Results

We picked 15 software bug datasets to compare the performance of several machine learning methods, including Naive Bayes, MLP, SVM, AdaBoost, Bagging, Decision Tree, Random Forest, J48, KNN, RBF, and K-means [19].We used the WEKA tool to carry out our experiments. The 10-fold cross validation test mode was used in the trials.

Table 3. Performance of different machine learning algorithms using cross-validation test mode grounded on mean absolute error

Datasets	**Naive Bayes**	**MLP**	**SVM**	**AdaBoost**	**Bagging**	**Decision Trees**	**Random Forest**	**J48**	**KNN**	**RBF**	**K-means**
AR1	0.17	0.11	0.08	0.12	0.13	0.12	0.13	0.13	0.32	0.13	0.11
AR6	0.17	0.19	0.13	0.22	0.24	0.25	0.22	0.23	0.25	0.22	0.17
CM1	0.16	0.16	0.10	0.16	0.16	0.20	0.16	0.17	0.16	0.17	0.14
JM1	0.19	0.27	0.18	0.27	0.25	0.35	0.25	0.26	0.33	0.28	0.23
KC1	0.18	0.21	0.15	0.22	0.20	0.29	0.19	0.20	0.18	0.23	0.17
KC2	0.16	0.22	0.17	0.22	0.22	0.29	0.22	0.23	0.21	0.23	0.21
KC3	0.15	0.12	0.09	0.14	0.14	0.17	0.14	0.13	0.39	0.15	0.12
MC1	0.06	0.01	0.01	0.01	0.01	0.03	0.01	0.01	0.31	0.01	0.01
MC2	0.27	0.32	0.28	0.39	0.37	0.40	0.35	0.32	0.35	0.41	0.31
MW1	0.16	0.11	0.08	0.12	0.12	0.15	0.12	0.12	0.18	0.12	0.13
PC1	0.11	0.11	0.07	0.11	0.10	0.14	0.09	0.10	0.12	0.12	0.08
PC2	0.03	0.01	0.00	0.01	0.01	0.02	0.01	0.01	0.18	0.01	0.01
PC3	0.51	0.14	0.10	0.16	0.15	0.21	0.15	0.15	0.36	0.18	0.13
PC4	0.14	0.12	0.11	0.15	0.14	0.16	0.14	0.12	0.43	0.20	0.13
PC5	0.04	0.03	0.03	0.04	0.03	0.06	0.03	0.03	0.33	0.05	0.03
Mean	0.16	0.14	0.10	0.15	0.15	0.18	0.14	0.14	0.27	0.16	0.13

Table 4. Performance of several machine learning algorithms that use cross-validation test mode grounded on the F-measure

	Supervised learning								**Unsupervised learning**		
Datasets	**Naïve Bayes**	**MLP**	**SVM**	**AdaBoost**	**Bagging**	**Decision Trees**	**Random Forest**	**J48**	**KNN**	**RBF**	**K-means**
AR1	0.90	0.94	0.96	0.95	0.96	0.94	0.96	0.95	0.79	0.95	0.94
AR6	0.90	0.91	0.93	0.90	0.92	0.90	0.92	0.90	0.84	0.92	0.90
CM1	0.91	0.94	0.95	0.95	0.95	0.94	0.94	0.94	0.91	0.95	0.93
JM1	0.89	0.90	0.90	0.90	0.90	0.90	0.90	0.88	0.80	0.90	0.86
KC1	0.90	0.92	0.92	0.91	0.92	0.92	0.92	0.91	0.89	0.92	0.91
KC2	0.90	0.90	0.90	0.88	0.90	0.89	0.89	0.88	0.86	0.90	0.88
KC3	0.91	0.94	0.95	0.95	0.95	0.95	0.94	0.94	0.72	0.95	0.93
MC1	0.97	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.81	1.00	1.00
MC2	0.82	0.78	0.82	0.80	0.81	0.77	0.80	0.78	0.76	0.81	0.77
MW1	0.90	0.95	0.96	0.95	0.96	0.95	0.95	0.95	0.89	0.96	0.93
PC1	0.94	0.97	0.96	0.96	0.97	0.97	0.97	0.97	0.94	0.96	0.96
PC2	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.90	1.00	1.00
PC3	0.60	0.94	0.95	0.95	0.94	0.95	0.94	0.94	0.77	0.95	0.93
PC4	0.92	0.94	0.94	0.94	0.94	0.93	0.94	0.93	0.72	0.93	0.92
PC5	0.98	0.99	0.99	0.98	0.99	0.98	0.99	0.99	0.80	0.99	0.99
Mean	0.89	0.93	0.942	0.93	0.94	0.93	0.93	0.93	0.82	0.93	0.92

Experiment procedure:

Input:

i) Datasets from the software bug repository.
D = {AR1, AR6, CM1, JM1, KC1, KC2, KC3, MW1, PC1, PC2, PC3, PC4, PC5}

ii) Selected machine learning techniques.
M = {Nayes Bayes, MLP, SVM, AdaBoost, Bagging, Decision Tree, Random Forest, J48, KNN, RBF, and K-means}

Data pre-process:

a) Apply Replace missing values to D
b) Apply Discretize to D
Test Model Cross Validation (10 folds):
for each D do for each M do
Perform cross-validation using 10-folds end for
Select accuracy
Select Mean Absolute Error Select F-measure end for

Output:

a) Accuracy
b) Mean Absolute Error
c) F Measure

IV. EXPERIMENT RESULTS

Tables 2, 3, and 4 illustrate the findings of the experiment. Three criteria were chosen to compare them: accuracy, mean absolute error, and F-measure. To compare the selected algorithms, the mean of all datasets was calculated, and the results are displayed in Figures 1–3[20].

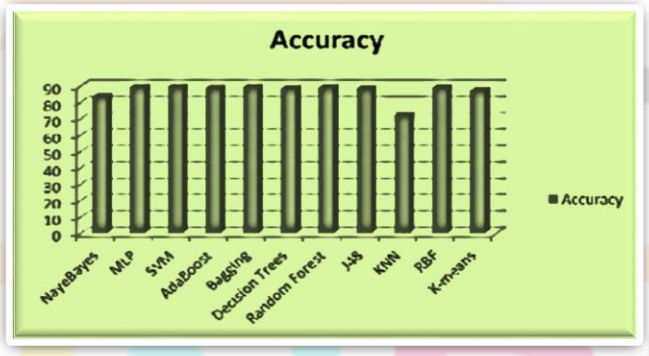

Figure 1. Accuracy results for chosen machine learning algorithms

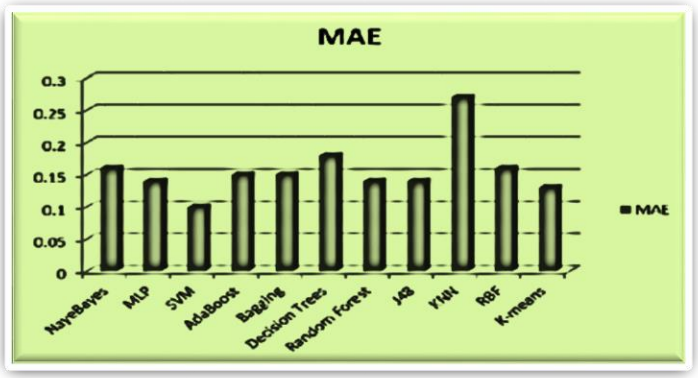

Figure 2. MAE outcomes for chosen machine learning algorithms.

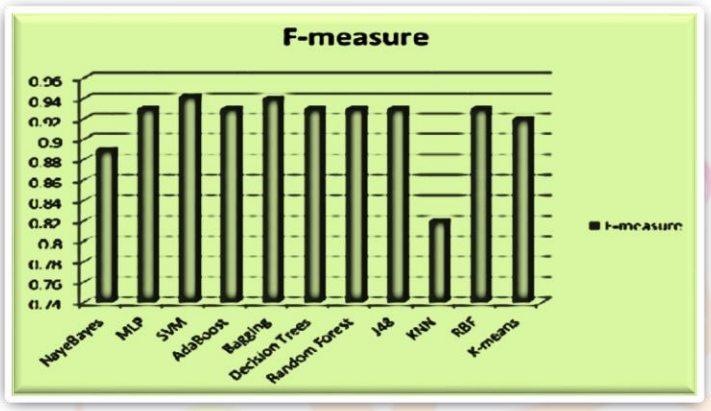

Figure 3: F-measure findings across various machine learning algorithms.

V. DISCUSSION & CONCLUSION

Tables 2, 3, and 4 exhibit the accuracy, F-measure, and MAE values from various datasets for different algorithms [21].The following conclusions were taken from the experiment results.The Naive Bayes classifier for software bug classification achieved an average accuracy of 83.47 across multiple datasets.It performed exceptionally well on the datasets MC1, PC2, and PC5, with accuracy rates over 95%.The poorest performance was obtained on the dataset PC3, which had an accuracy of less than 50%. MLP also performed well on MC1 and PC2, with an overall accuracy of 89.14% across various datasets.SVM with bagging outperformed other machine learning algorithms, achieving an overall accuracy of approximately 89%.Adaboost achieved an accuracy of 88.59, bagging achieved an accuracy of 89.386, decision trees achieved an accuracy of roughly 88.47, Random Forest achieved an accuracy of 89.08, J48 achieved an accuracy of 88.33, and in the case of unsupervised learning, KNN achieved 71.99, RBF reached 88.87, and K-means achieved 87.29. MLP, SVM, and bagging performed well on all of the datasets when compared to other machine learning algorithms.The KNN algorithm produced the lowest accuracy.The best MAE achieved by the SVM approach was 0.10 on multiple datasets, with 0.00 MAE for the PC2 dataset. The KNN algorithm had the worst MAE at 0.27. K-means, MLP, Random Forest, and J48 all achieved improved MAEs of around 0.14. In the case of the F-measure, higher is better. SVM and bagging algorithms achieved a higher F-measure of around 0.94. The worst F-measure achieved by the KNN technique was 0.82 across many datasets.

VI. CHALLENGES AND FUTURE WORK

The subject of software bug finding by machine knowledge presents various obstacles that require further investigation and invention [22].Imbalanced datasets are a major concern for perambulator instances, as they are usually lower than non-buggy bones. This makes oversampling, undersampling, and synthetic data products sensitive, much like SMOTE [23]. Another difficulty is that machine knowledge models may be used on a wide range of software systems with different rendering styles, languages, and methods [24].Developing lightweight and efficient models for real-time bug detection within integrated development environments (IDEs) is another critical area for growth. Furthermore, cold-bred approaches that mix the benefits of machine knowledge with classic static and dynamic logical styles provide a viable option for developing more robust results [25].

REFERENCES

[1] Marçal ,I., & Garcia, R. E. (2023). A comprehensible analysis of the efficacy of Ensemble Models for Bug Prediction. arXiv preprint arXiv:2310.12133.
[2] Saharudin, S. N. A., 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.
[3] Shailee, N. M., Alam, A., Ahmed, T., & Nur, K. (2024). Software Bug Prediction using Machine Learning on JM1 Dataset.

[4] Pusarla, S., Peruri, G. S., & Yalavarthi, M. (2024). An empirically based object-oriented testing using Machine learning. EAI Endorsed Transactions on Internet of Things.
[5] Ferenc, R., Siket, I., Hegedűs, P., & Rajkó, R. (2020). Employing Partial Least Squares Regression with Discriminant Analysis for Bug Prediction. arXiv preprint arXiv:2011.01214.
[6] Li, L., Lessmann, S., & Baesens, B. (2019). Evaluating software defect prediction performance: an updated benchmarking study. arXiv preprint arXiv:1901.01726.
[7] Zhang, H., & Zhang, X. (2021). Deep learning-based software defect prediction with class imbalance and missing data handling. IEEE Transactions on Software Engineering, 47(5), 930-950.
[8] Wang, S., Liu, T., & Tan, H. B. K. (2022). A survey on deep learning for software defect prediction: Advances, challenges, and opportunities. Journal of Systems and Software, 183, 111089.
[9] Chen, Z., & Li, Y. (2021). Cross-project defect prediction via transfer learning: A systematic review. Information and Software Technology, 133, 106497.
[10] Yang, X., Lo, D., & Sun, J. (2023). TLEL: A two-layer ensemble learning approach for just-in-time defect prediction. IEEE Transactions on Software Engineering, 49(1), 1-17.
[11] Huang, Q., & Li, B. (2022). An empirical study on the impact of data imbalance in software defect prediction. Empirical Software Engineering, 27(3), 1-30.
[12] Liu, Y., & Shen, J. (2021). A novel ensemble learning approach for software defect prediction. Applied Soft Computing, 101, 107026.
[13] Kamei, Y., & Matsumoto, S. (2020). A large-scale empirical study of just-in-time quality assurance. Empirical Software Engineering, 25(3), 2071-2106.
[14] Zhou, Y., & Jiang, Y. (2021). A comprehensive analysis of defect prediction: A meta-analysis approach. IEEE Transactions on Software Engineering, 47(4), 720-740.
[15] Nam, J., & Kim, S. (2020). Heterogeneous defect prediction with mixed types of static code attributes. IEEE Transactions on Software Engineering, 46(12), 1373-1386.
[16] Fang, L., & Zhang, M. (2023). Comparative evaluation of machine learning models for software defect prediction. Software: Practice and Experience, 53(6), 1423-1438.
[17] Giray, G., Köksal, Ö., & Tekinerdogan, B. (2022). On the use of deep learning in software defect prediction. Expert Systems with Applications, 206, 117891.
[18] Xia, X., & Lo, D. (2021). A systematic review of deep learning for software defect prediction. ACM Computing Surveys, 54(5), 1-34.
[19] Jiang, H., & Hassan, A. (2023). Understanding and improving software defect prediction models with explainable AI. Journal of Software: Evolution and Process, 35(3), e2451.
[20] Saiqa Aleem, Luiz Fernando Capretz, and Faheem Ahmed, "Comparative Performance Analysis of Machine Learning Techniques for Software Bug Detection," Computer Science & Information Technology (CS & IT), pp. 71-79, 2015. DOI: 10.5121/csit.2015.50108.
[21] Yin, Z., & Shen, J. (2022). Hybrid approaches for software defect prediction: A comparative study. Information and Software Technology, 140, 106758.
[22] Yang, Y., & Wu, S. (2023). A benchmark comparison of ensemble learning techniques for software defect prediction. Software Quality Journal, 31(2), 345-370.
[23] Gupta, R., & Reddy, P. (2024). An empirical study of ensemble learning techniques for software defect prediction. Information and Software Technology, 164, 107187.
[24] Ali, M., & Shahzad, T. (2024). Enhancing software defect prediction: A framework with improved feature selection and ensemble machine learning. Applied Sciences, 12(9), 4577.
[25] Reddy, K. S., & Rajesh, B. (2024). Software defect prediction using an intelligent ensemble-based model. International Journal of Communication Networks and Information Security, 16(5), 60-68.