Grenze International Journal of Engineering and Technology, June Issue

A Hybrid Ensemble Method for Accurate Software Defect Prediction

Thota Sai Lalith Prasad[1], Sarangi Jyoshnapriya[2] and Jiguri Lakshmi Vydehi[3]
[1,3]Assistant Professor-Artificial Intelligence & Data Science Department, Vignan Institute of Technology and Sciences, Telangana, India.
Email: lalithresearch15@gmail.com, j.l.vydehi@gmaill.com
[2]PG Scholar-Artificial Intelligence & Data Science Department, Vignan Institute of Technology and Sciences, Telangana, India
Email: vogumala2000@gmail.com

***Abstract*— The software defect prediction applies in improving the quality of the software delivered and the cost of testing the software as the testers will only test the modules which are defected. The Synthetic Minority Over-Sampling Technique (SMOTE) and Particle Swarm Optimization (PSO) are used in this classification unlike CM1 or JM1 because the researchers of such issues as the unequal representation of the classes or the most effective selection of features should be allowed. In case of a fused Stacking Classifier with Decision Tree, the Random Forest, and the Light GBM models, which were also rather effective and applied to all the datasets, respectively. The additional proposals are provided to render the proposed approach more accurate due to the assistance of the ensemble learning that presupposes the synthesis of the most effective aspects of multiple classifiers. The proposed model has also undergone the relevant tests and it is determined that it was a good and sound model. These results also indicate that it has a high degree of locating the compromised software modules as well as the software modules prediction. This execution is aimed to have an advantage to the software fault forecasting, in such a manner, that it is an inexpensive and swift approach of detecting errors at the initial stage. This will lead to the fact that the usage and quality of the testing resources will be improved.**

***Index Terms*— Machine learning, software defect prediction, ensemble classification, heterogeneous classifiers, random forest, support vector machine, naïve Bayes.**

I. INTRODUCTION

The computer program business is very strong in the contemporary globalized world to realize improvements in every aspect. The consequence of high rate of globalization is the transformation of the world into a global village. It is not only that the software applications are mundane, it is that they are actually unparalleled in what they mean, in both the life and business, in their day-to-day operations and infrastructural necessities [1, 2]. Online technology is becoming a more relevant mode of communication between individuals and companies to conduct business, work and other business and meaningful remote transactions [3, 4, 5]. It is more reliant to the software systems in the COVID-19 pandemic. The modern world is redefining its presence in every nook and thus a good software has emerged the ultimate concern of the business.

Grenze ID: 01.GIJET.12.2.332_22

Table I Software Development Life Cycle shows the great extent of cooperation between the software development and QA (Gibson, 2004). The first facet of commencing the processing of the code to practice is the development team. The code would then be provided to the QA team which would also put it to test. At this stage of the consideration, the software issues or other bugs should be found and reported. The developers get feedback and ensure corrections in areas that are referred to as having flaws. This will continue to the creation of a good software product that is bug-free [6, 7]. The flow of quality assurance taken into account including this creation has been represented in SDLC as illustrated in Fig. 1 below. Such a process is mainly aimed at making the software of standard, useful and reliable to be published to the end user.

Nonetheless, at times it is not that easy to separate the software that is bug-free. The determination as practical is that there are three critical factors, which predetermine the efficiency of the software quality assurance, time, money and qualified employees. The need to identify cars to pass through the test procedures in software delivery is economical and quick in software delivery velocity increases with the propensity of having software delivery pace influx. The companies are also at a constant pressure of providing quality software at a reasonable cost within a reasonable time by fitting the existing resources they have to it in the most effective manner feasible to them [8]. As such it has added more to the individuals interests in the form of automated bug prediction that has expedited the QA process.

The potential solution to the problems has been found. In SDP, the machine learning (ML) is utilized to consider the past information on software, and carry out the assumption on the likeliness of the bugs resurfacing in the new modules of software. The measurement of the various software such as the code complexities, size, and data on the past bugs is one of the possible places where SDP models can be utilized in helping teams to identify bugs even before they go to the testing stage [9]. It is a proactive measure that will help business organisations to focus on testing, minimize a chance of bugs escaping when it is launched and later on, enhance the quality of the software and efficiency of resource use [10]. Due to this fact SDP is becoming popular as a high-potent tool of software development. It helps the companies respond to the growing need and request in the world which is interdependent and more complex.

II. Related Work

Specifically, Software Defect Prediction (SDP) deserves specific attention, as there is a strong need to exhaust the avenue of resources such as time, money and man-power to the last resource available in the process of software creation. The SDP Theory is also expected to identify the failed software's in the very early part of the product development. This allows cutting down the period of time taken to carry out testing and repair. The subsequent analogous research on the majority of machine learning estimations and conglomerative methods has acted as the concept behind the development of further superior and enhanced and consistent program defect estimates in the previous few decades [11]. Under literature review, the author talks about the critical concepts and techniques, which have been applied to this field. It is one-sided in the selection of the features, unbalance of the classes and the ensemble learning [12].

It is possible to make neural networks successful to detect a complex pattern of software measurements. This is what made them advantageous in modeling in which the failures of the software are expected. M. S. The [13] conducted one such experiment with a purpose to find out how the neural networks and the feature selection methods can be used in a case when predicting computer bugs. The paper has pointed to the necessity of establishing the most important characteristics of the information considering the goal of producing the predictions as close as possible. The choice of the features assists in reducing dimensions of the data that leads to the acceleration of the training time and overfitting. This helps in a development of the prediction of the defects in the long run. Regarding the experiment performed by Alkhasawneh, in case they are combined with the successful methods of features selections, the neural networks can be more specific than the conventional machine learning classifiers.

The other article that compared the determination of defects on the basis of the Least-Squares Support Vector Machines (LSSVM) was carried by [14]. It involved the process of making a decision on the feature to select the most significant attributes in the data that produced the most significant change, as far as the prediction process was concerned. The authors also demonstrate that such a combination has greater predictive power of prediction implying that the selection of the proper features is a critical step in modeling an even more machine learning can be a predictor of defects.

One of the largest issues at SDP is the problems of the issues of incongruencies of classes. Bad programs are generally small in contrast to non-bad programs. Such imbalance may lead to bias models and this can be explained by the fact that the forecast of majority class is inaccurate. When handling this, there have been other

options of sampling data considered, including oversampling, under sampling data field and faking data field like SMOTE [15].

To meet the objective of class disparity in SDP, [15] adhered to the empirical research of the varied data sampling techniques. The authors in question have chosen more sophisticated sampling procedures such as SMOTE and less sophisticated sampling procedures such as random oversampling and under sampling. The findings indicated that the sampling strategies that incorporated the usage of SMOTE are effective since they generated manmade samples of the minority group. This has made the data set a homogenous collection of data and model more accurate. Moreover, the research asserts that the need to employ data sampling methodology, machine learning settings, the Random Forest and the SVM, could help to streamline the task of the disclosure of the malfunctioning software modules array to a considerable extent.

III. Materials and Methods

They seek to obtain an approximation value on whether defects can be rightly predicted by the use of the system. It is chosen on to one more definite and convincing. The other one of such classifiers is an Adaptive Voting Classifier that is an amalgamation of the best, the Rand Forest classifier, the SVM, the naive Bayes and the MLP. It also exploits the potential of the various learning styles and brings about the potential of detecting bugs with the help of the Stacking Classifier that was built on the basis of the framework of the Decision Tree, the Random Forest, and the Light GBM designs. Such approaches as Synthetic Minority Over-sampling Technique (SMOTE) that are directed towards the solution of imbalance within the classes have been done. In its turn, PSO would be used in the selection of the most efficient features, and ensuring that the model is effective and efficient. It will based on this model to scan the damaged modules and prioritize on quality software development to save time that would be spent in the lengthy process of testing.

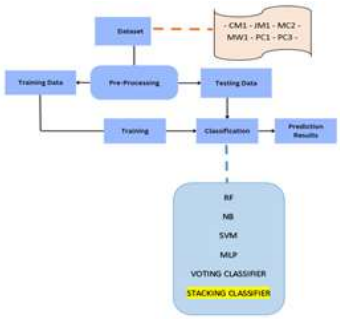

Figure 1. Proposed Architecture

The proposed architecture, as presented in Fig. 1 has several steps such as dataset collection, preprocessing, training and testing. RF, NB, SVM and MLP are used as classification models on the processed data. Predictions are combined with a voting classifier and then a stacking classifier is added to enhance the overall performance. Lastly, the system produces valid prediction outcomes of the integrated model approach.

The picture presents one of the instances of the machine learning application to the classification issues. It starts by a number of data which is split into trading and training set. Training data is pre-processed in such a way that they could be availed at the time of model training. The other machine learning algorithms are then trained using the previous training data that has been cleaned. These models get fed with the training data and after that are used to categorize the information in a test data and their conjecture of the real names is done to ascertain their efficiency. The other methods such as Voting Classifier and Stacking Classifier can also be classified as the ensemble methods whereby, according to them, the outputs of combining more than one model are added together and they may lead to the enhancement of the overall performance.

Dataset

One must gather the previous software defect files which would be the first stage of the training process. Most popular data sets benchmark that has been utilized in this paper can be located in NASA Metrics Data Program (MDP) repository. They are CM1, JM1, MC2, MW1, PC1, PC3 and PC4 datasets. Majority of the research in prediction of elements of software bugs is based on them in a way that the findings are comparative and can be contrasted with that of other studies conducted. One software component and one software module are the dataset and instance respectively. The modules that take over the different software quality characteristics e.g.

LOC_COMMENT, LOC_TOTAL, CALL_PAIRS, HALSTEAD_LENGTH and HALSTEAD_CONSTANT are distributed through SDLC. The dependent attributes are considered to calculate the forecasts and the independent attributes are employed to state whether a module is not broken or broken [25].

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	9.0	5.0	3.0	2.0	2
1	19.0	3.0	1.0	2.0	0
2	0.0	9.0	0.0	0.0	0
3	2.0	15.0	2.0	1.0	9
4	5.0	5.0	1.0	0.0	0

Figure 2. Dataset for CM1

As indicated in Fig. 2, CM1 dataset includes different metrics of software including LOC (Lines of Code), number of branches, call pairs, and code/comment. The input parameters in training and evaluation of the prediction models are these features. The data set contains crucial data needed to analyze software defects and enhance the model function.

	LOC_BLANK	BRANCH_COUNT	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CYCLOMAT
0	1.0	7.0	0.0	0.0	
1	5.0	37.0	0.0	6.0	
2	2.0	1.0	0.0	0.0	
3	16.0	1.0	0.0	0.0	
4	0.0	7.0	0.0	0.0	

Figure 3. Datasets for JM1

Fig. 3 shows that the JM1 dataset has a number of software measures like LOC (Lines of Code), number of branches, LOC code and comments, and cyclomatic complexity. These characteristics are used in training and testing machine learning models in defect prediction. The data can be used to examine the quality of the code and find possible modules that contain faults.

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	16.0	19.0	0.0	0.0	22
1	9.0	13.0	2.0	0.0	14
2	2.0	3.0	0.0	0.0	0
3	35.0	97.0	3.0	0.0	51
4	1.0	3.0	1.0	1.0	0

Figure 1. Datasets for MC2

The MC2 dataset, as presented in Fig. 4, is a set of software measures which includes LOC (Lines of Code), number of branches, call pairs, and LOC code and code comments. The features are the input features that are used to train and evaluate defect prediction models. The data enables the research of software quality and assists in effectively determining the modules that are likely to have defects.

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	5.0	7.0	13.0	0.0	7
1	5.0	21.0	3.0	1.0	2
2	2.0	5.0	0.0	0.0	4
3	3.0	5.0	3.0	0.0	15
4	3.0	5.0	2.0	0.0	5

Figure 5. Datasets for MW1

As demonstrated in Fig. 5, MW1 dataset also has software metrics like LOC (Lines of Code), the number of branches, the number of call pairs, and LOC code and comments. The features are applied in training and testing machine learning models of the prediction of software defects. The data set can help to assess the performance of the model and which modules have higher chances to have defects.

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	1.0	3.0	2.0	0.0	0
1	2.0	5.0	2.0	1.0	3
2	0.0	3.0	1.0	0.0	0
3	2.0	3.0	2.0	0.0	0
4	19.0	17.0	13.0	0.0	0

Figure 6. Datasets for PC1

As indicated in Fig. 6, PC1 dataset has software measure of LOC (Lines of Code), number of branches, number of call pairs, and LOC code and comments. Input features are these attributes, which are used to train and evaluate defect prediction models. The data set is useful to characterize the quality of software and track down the fault-prone modules.

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	16.0	13.0	1.0	6.0	11
1	2.0	7.0	0.0	0.0	7
2	1.0	13.0	5.0	0.0	0
3	8.0	3.0	1.0	0.0	1
4	1.0	5.0	2.0	1.0	1

Figure 7. Datasets for PC3

According to Fig. 7, PC3 dataset has software metrics, including LOC (Lines of Code), branch count, call pairs, and LOC code and comments. The use of these features is in training and testing machine learning models in the prediction of software defects. The data can be utilized to analyze the quality of the code and determine the possible modules with defects.

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	8.0	1.0	0.0	0.0	6
1	1.0	7.0	3.0	0.0	0
2	33.0	29.0	7.0	9.0	20
3	1.0	19.0	0.0	17.0	0
4	7.0	13.0	5.0	9.0	9

Figure 8. Datasets for PC4

As Fig. 8 illustrates, PC4 dataset comprises software measures of LOC (Lines of Code), number of branches, number of call pairs and LOC code and comments. Training and assessing defect prediction models are based on these attributes as input features. The data can be used to analyze the software quality effectively and help in telling about the modules likely to contain bugs.

Algorithms

Random Forest: The random forests is also an ensemble type of a learning since it also teaches many decision trees and ways of estimating the outcome and averaging it to get a more accurate yet less-overfitting modified model. It has been applied in this effort to calculate the software errors by the fact that the outcome of two or even more trees is embraced to offer a less intermittent and dependable nomenclature of whether a module is not broken or there are in baselines that suits to identify the highly desired hyperplane as well as classify the classes in a multi-dimensional region. SVM is used in this project in such a manner that, sub-categories of the software modules can be classified based on the characteristics of quality in them. It is quite able to use it to distinguish a defective and non-defective module.

IV. Results and Discussions

Accuracy: Accuracy of a test can be seen as the capacity through which a test is able to capture the difference between a sick and a healthy person. The validity of the test can be established based on the percent of the number of the true positives and false negatives of the number of all the cases under test. In terms of math, this is:

$$Accuracy=\frac{TP+TN}{TP+FP+TN+FN} (4)$$

Precision: Precision denotes the number of the cases or samples that is deemed to be positive. Precision =.023/.00004 The formula used to compute Precision is shown below:

$$Precision=\frac{TruePositive}{TruePositive+FalsePositive} (5)$$

Recall: The concept of recall is used in machine learning to establish the extent to which a model can identify all the important examples of one of the classes. It quantifies the instances of predicted correctly positives to the instances of positive ones to the research and can be utilized to determine how effectively a model can recover the instances of a group.

$$Recall=\frac{TP}{TP+FN}(6)$$

F1-Score: F1 score is the accuracy of a learning model. The model accuracy and model recall scores were added. The amount of accuracy illustrates the frequency of it is correct on entire data set of a model.

$$F1Score=2*\frac{RecallXPrecision}{Recall+Precision}*100(7)$$

Sensitivity: Sensitivity as the measure with respect to a test and instruments can be revealed as ability of the instruments to actualize a situation in a person. It is defined as the proportion of individuals who are correctly identified as positive among all individuals who actually have the disease.

$$Specificity=\frac{TP}{(TP+FN)}(8)$$

Specificity: It can be determined by the percentage of numbers of people who test negative to a disease/ total number of people who do not develop such an illness. It is concerned with the individuals that tested negative as well as the individuals that tested positive and they did not have the disease.

$$Specificity=\frac{TN}{(TN+FP)}(9)$$

A curve that brings out the issue of classification that was resolved at various levels of benchmarks.The resulting difference between classes that the model is good at discriminating is the difference which is referred to as the AUC; the greater the AUC the finer the model.

$$AUC=\sum_{i=1}^{n-1}(FPR_{i+1}\text{-}FPR_i)\cdot\frac{TPR_{i+1}+TPR_i}{2}(10)$$

Research performance of each algorithm according to the studies of each in respect to F1-score, AUC Score, Specificity, Sensitivity, and accuracy, precision and recall. Using Stacking Classifier, all other algorithms are inferior in both of the datasets. The tables also indicate the comparisons of the measures of the other algorithms.

TABLE I: PERFORMANCE EVALUATION METRICS FOR CM1

ML Model	Accuracy	Precision	Recall	F1 Score	AUC Score	Specificity	Sensitivity
Random Forest	0.918	0.921	0.918	0.918	1.000	0.918	0.918
Support Vector Machine	0.713	0.719	0.713	0.714	0.833	0.714	0.713
Naive Bayes	0.719	0.765	0.719	0.726	0.783	0.721	0.719
MLP Classifier	0.860	0.867	0.860	0.860	0.961	0.859	0.860
Adaptive Voting Classifier	0.865	0.866	0.865	0.866	0.971	0.865	0.865
Stacking Classifier	0.924	0.924	0.924	0.924	1.000	0.924	0.924

Table 1 displays the performance analysis of different machine learning models on the CM1 dataset in terms of accuracy, precision, recall, F1-score, AUC score, specificity, and sensitivity. The stacking classifier has the best performance with the accuracy of 0.924 and the AUC score of 1.000. The adaptive voting classifier and MLP classifier perform well as well, whereas SVM and Naive Bayes perform relatively poor. This shows that ensemble techniques are more effective in predicting defects.

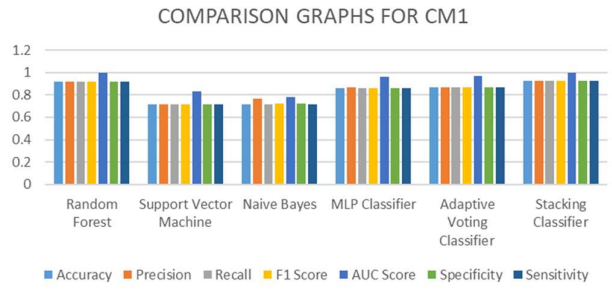

Fig 9. Graph 1. Comparison Graphs for CM1

As shown in Graph. 1, the various classifiers are compared on the CM1 dataset by the use of performance measures like accuracy, precision, recall, F1-score, AUC score, specificity, and sensitivity. Stacking classifier has exhibited higher performance in most of the evaluation measures in comparison to other models such as random forest, SVM, Naive Bayes, and MLP. This shows that ensemble learning methods are effective in enhancing prediction and the general model performance.

TABLE II: PERFORMANCE EVALUATION METRICS FOR JM1

ML Model	Accuracy	Precision	Recall	F1 Score	AUC Score	Specificity	Sensitivity
Random Forest	0.840	0.840	0.840	0.840	1.000	0.840	0.840
Support Vector Machine	0.588	0.786	0.588	0.633	0.643	0.588	0.588
Naive Bayes	0.582	0.825	0.582	0.639	0.624	0.582	0.582
MLP Classifier	0.608	0.622	0.608	0.611	0.613	0.608	0.608
Adaptive Voting Classifier	0.661	0.735	0.661	0.674	0.851	0.661	0.661
Stacking Classifier	0.836	0.837	0.836	0.836	1.000	0.836	0.836

The results of various machine learning models on the JM1 dataset are measured with accuracy, precision, recall, F1-score, AUC score, specificity, sensitivity, as indicated in Table 2. Random Forest and Stacking Classifier have the best results with an accuracy of 0.840 and 0.836 respectively, and an AUC of 1.000. The performance of Adaptive Voting is moderate, whereas the results of SVM, Naïve Bayes as well as MLP classifiers are relatively low. This shows that the ensemble and tree-based models work well with the JM1 data.

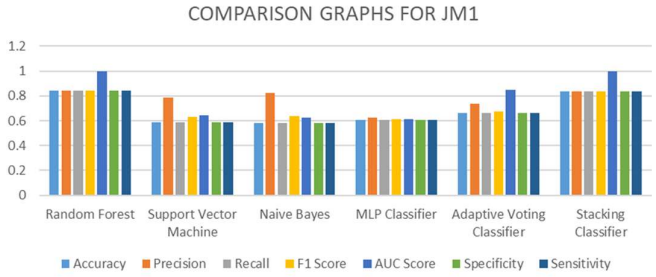

Fig 10. Graph 2. Comparison Graphs for JM1

As shown in Graph. 2, a comparison of different classifiers on the JM1 dataset is shown in the form of performance indicators like accuracy, precision, recall and F1-score, AUC score, specificity and sensitivity. The stacking classifier has the best performance in most metrics among all models, and then the adaptive voting classifier. This underlines the usefulness of ensemble methods to improve model accuracy and reliability in predicting defects.

V. CONCLUSION

Software defect prediction is determined so as to detect the malfunctioning modules in its earlier stages. The most suspicious to bugs modules in that case can be only experimented. Appropriate defect predictive model is used to cut high percentages of the software development cost through efficient use of the facilities that may be in our disposal as part of the quality assurance attempts. Such sophisticated processes as SMOTE will be used to manage the issue of unevenly distributed classes, and, PSO will be used to choose the features to provide the fact that the training data were equal and applicable. One of all those methods that were pursued was the Stacking Classifier which proved to be the best and advantageous in all data sets. It managed to do this through making predictions on the different learning patterns so as to bring the prediction nearer. The need arises to develop defective modules in an appropriate way, locate them within a shorter time and help in the development of good software using small finances and time. The fact is explained by such a hybrid approach and even the high-performance ensemble learning.

One can apply deep learning models in the future i.e. CNN and LSTM to detect more delicate patterns in the data on software bugs, and utilise them along with other deep learning models. The predictions of the explainable AI strategies would be more conveniently comprehended and this would be of benefit to the creators with an attempt to rectify the root cause of bugs. The approach would entail applying the method to bigger volumes of the real world and software systems that the approach is intended to be applied to raise the scalability and generalizability of the approach. One can also multiply the results of the predictability of defects with the more

developed sampling methods and even with the optimization methods that will provide the various fields with cheaper and quality software development.

REFERENCES

[1] I. Sommerville, Software Engineering, 10th ed. Pearson, 2016.
[2] R. S. Pressman and B. R. Maxim, Software Engineering: A Practitioner's Approach, 8th ed. McGraw-Hill, 2014.
[3] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, 2005.
[4] M. Fowler, Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.
[5] A. Tanenbaum and H. Bos, Modern Operating Systems, 4th ed. Pearson, 2015.
[6] G. Gibson, "Software development life cycle models and methodologies," IEEE Software, vol. 21, no. 5, pp. 45–50, 2004.
[7] P. Jalote, An Integrated Approach to Software Engineering, 3rd ed. Springer, 2005.
[8] L. Briand, W. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," IEEE Transactions on Software Engineering, vol. 28, no. 7, pp. 706–720, 2002.
[9] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 2–13, 2007.
[10] Q. Song, Z. Jia, M. Shepperd, et al., "A general software defect-proneness prediction framework," IEEE Transactions on Software Engineering, vol. 37, no. 3, pp. 356–370, 2011.
[11] M. S. Alkhasawneh, "Software defect prediction using neural networks and feature selection methods," International Journal of Advanced Computer Science and Applications, vol. 9, no. 5, pp. 123–130, 2018.
[12] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," Information and Software Technology, vol. 55, no. 8, pp. 1397–1418, 2013.
[13] X. Yu, S. Wang, and Z. Li, "Feature selection for software defect prediction using machine learning techniques," Journal of Systems and Software, vol. 138, pp. 121–136, 2018.
[14] T. F. Husin and M. R. Pribadi, "Software defect prediction using least squares support vector machines," International Journal of Electrical and Computer Engineering, vol. 10, no. 2, pp. 1452–1459, 2020.
[15] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.