International Journal of Advanced Research and Multidisciplinary Trends (IJARMT)
An International Open Access, Peer-Reviewed Refereed Journal
Impact Factor: 6.4 Website: https://ijarmt.com ISSN No.: 3048-9458

Optimization Accuracy of Software Defect Prediction using Random Forest Machine Learning Algorithm

Swati Priya
M. Tech. Scholar, Department of Computer Science and Engineering, SORT, People's University, Bhopal, India
Mr. Sudhir Goswami
Assistant Professor, Department of Computer Science and Engineering, SORT, People's University, Bhopal, India

Abstract: - Software defect prediction is a critical task in software engineering that aims to identify faulty components early in the development lifecycle, thereby reducing cost and improving software quality. This project explores the effectiveness of the Random Forest machine learning algorithm for predicting software defects, with a particular focus on optimizing model accuracy. The Random Forest algorithm, known for its robustness and ability to handle high-dimensional data, is applied to benchmark software defect datasets. The study involves extensive preprocessing, including feature selection and normalization, followed by hyperparameter tuning to enhance prediction performance. Evaluation metrics such as accuracy, precision, recall, F1-score, and AUC-ROC are used to assess the model's effectiveness. Experimental results demonstrate that the optimized Random Forest model achieves high predictive accuracy and outperforms several baseline models. This work highlights the potential of ensemble learning methods, particularly Random Forest, as a reliable approach for software defect prediction, aiding developers in creating more reliable and maintainable software systems.

Keywords: - Software Defects, Accuracy, Precision, Recall, Random Forest (RF), Logistic Regression (LR)

I. INTRODUCTION

SDP is essential to guaranteeing the dependability and caliber of software systems. Finding flawed modules early in the development process is essential to lowering maintenance costs and enhancing overall performance as software complexity rises. By identifying patterns in past software data, machine learning techniques have become powerful instruments for automating defect prediction. The Random Forest algorithm is unique among these methods because of its excellent accuracy, capacity to deal with unbalanced data, and resistance to overfitting. The goal of this research is to employ the Random Forest algorithm to maximize the accuracy of software defect prediction. The objective is to improve prediction performance and facilitate improved decision-making in software quality assurance by utilizing preprocessing approaches, feature selection, and hyperparameter tuning.

SDP employs software metrics (also referred to as software features or software attributes, such as lines of code (LOC), and change information to predict their defect-proneness to support software testing activities [3]. Software defects have become one of the main reasons for the failure of large engineering projects, leading to huge economic losses in the 21st century. In software quality, various defect prediction techniques have been proposed. Essentially, such techniques rely on different predictors, including source code metrics (e.g., coupling, cohesion, size). Software defect prediction can help to allocate testing resources efficiently through ranking software modules according to their defects. Existing software defect prediction models that are optimized to predict explicitly the number of defects in a software module might fail to give an accurate order because it is very difficult to predict the exact number of defects in a software module due to noisy data [4].

Each software defect is generated under different conditions and environments, and hence differs in its specific characteristics. A software defect may have an enormous negative effect upon software quality. Therefore, defect prediction is extremely essential in the field of software quality and software reliability. Defect prediction is comparatively a novel research area of software quality engineering [5]. Current defect prediction work focuses on

1) Estimating the number of defects remaining in software systems,
2) Discovering defect associations, and
3) Classifying the defect-proneness of software components, typically into two classes, defect-prone and not defect-prone.

The outcome of the forecast can be utilized as a crucial metric by the software developer to regulate the software development process. SDP empirical research frequently suffer from limited generalizations and have significant biases with regard to data quality [6, 7]. It is anticipated that defect predictors will lower the cost of supplying those software systems while also enhancing software quality. Since the creation of the Promise Repository in 2005, SDP research has expanded quickly. It enables researchers to create repeatable, comparable models across studies and provides a library of publicly available defect prediction data sets from actual projects. In order to create prediction models for SDP, a lot of research has been done so far on metrics that describe code modules and learning techniques [8].

II. SOFTWARE QUALITY AND SD PREDICTION

Software quality refers to the degree to which a software system meets specified requirements and user expectations, encompassing aspects such as functionality, reliability, efficiency, maintainability, and usability. High-quality software is essential for ensuring user satisfaction, reducing maintenance costs, and enhancing system performance.

Software Defect (SD) prediction is a vital process within software quality assurance. It aims to identify parts of a software system that are likely to contain defects or bugs before they manifest during deployment or use. Early prediction of software defects helps in prioritizing testing efforts, improving resource allocation, and reducing time-to-market.

Traditional software testing methods are often time-consuming and costly, especially when applied to large-scale software systems. As a result, data-driven approaches using machine learning have gained traction for automating and enhancing defect prediction. These methods leverage historical data, such as source code metrics, bug reports, and change logs, to train predictive models that can forecast the likelihood of defects in new or modified code.

Among various machine learning techniques, ensemble methods like Random Forest offer high accuracy and robustness in defect prediction tasks. By combining multiple decision trees, Random Forest reduces variance and improves generalization, making it suitable for complex and noisy software data. This project focuses on utilizing and optimizing Random Forest to improve the accuracy of software defect prediction, thereby contributing to the development of more reliable software systems.

2.2 Software Defect Prediction

The error in the algorithm or in the software program will not permit the software product to satisfy the user requirements and because of that, the user expectations and the software requirement standards are not maintained by the product and is also called as software defect. The error in the software sometimes produce unexpected outcome and cause software malfunctioning too. There are several ways to define the defects produced by the software: -

- The defect or the bug in the application may be caused or created due to some mistake of the programmer.
- If there is a deviation in the expected result produced by the software and it is not producing the result specified or defined in Specification document, then it is considered as a defect.
- Failing to satisfy the end user expectations is considered as defect in the software and it is mainly due to the bugs arising in the program or the methods used when the product is developed.

The foremost thing when developing a piece of software is to produce it with great eminence and with high excellence. Superiority of the software is measured by the degree to which the piece of code meets the requirements specified in the requirement specification document. [13].

III. PROPOSED METHODOLOGY

A system that is used to understand the concept and its environment using a simplified interpretation of the environment using model is called cognitive system. The step that we pass to construct the model is known as inductive learning. The Cognitive system is able to combine its experience by constructing new structures is patterns. The constructed model and pattern by cognitive system is called machine learning. Models that are described as predictive since it can be used to predict the output of a function (target function) for a given value in the function‘s domain while informative pattern are characterized only describes the portion of data.

Random Forest is an ensemble method based on principle of bagging. It uses decision trees as base classifiers. To generate each single tree in Random Forest, Breiman followed following steps: If the number of records in the training set is N, then N records are sampled at random but with replacement, from the original data; this is bootstrap sample.

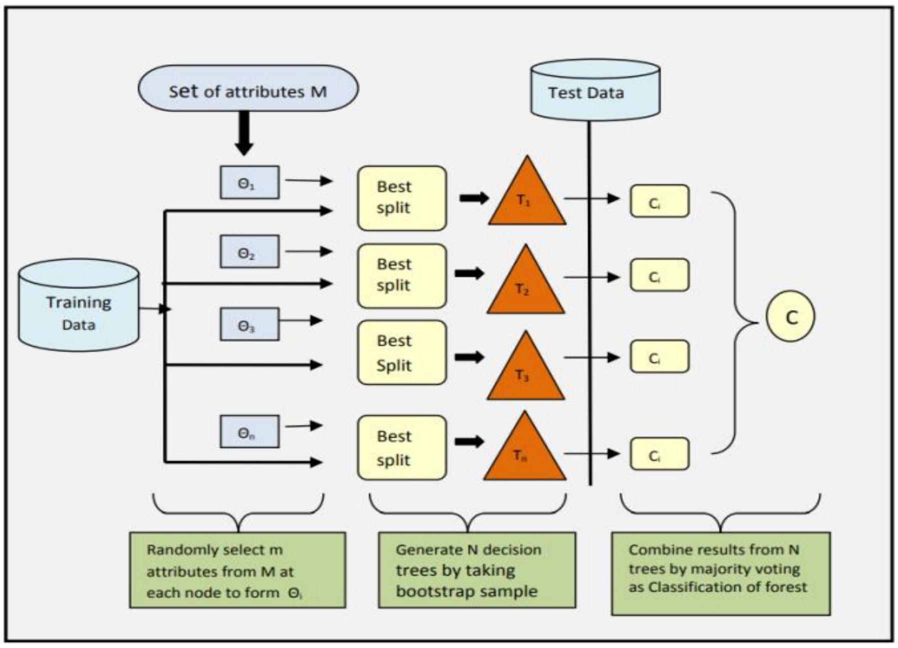

Set of attributes M
Test Data
Training Data
Θ₁
Θ₂
Θ₃
Θₙ
Best split
Best split
Best Split
Best split
T₁
T₂
T₃
Tₙ
Cᵢ
Cᵢ
Cᵢ
Cᵢ
C
Randomly select m attributes from M at each node to form Θᵢ
Generate N decision trees by taking bootstrap sample
Combine results from N trees by majority voting as Classification of forest

Figure 1: Random Forest classifier

This sample will be the training set for growing the tree. If there are M input variables, a number m << M is selected such that at each node, m variables are selected at random out of M and the best split on these m attributes is used to split the node. The value of m is held constant while the forest is growing. Each tree is grown to the largest possible extent. There is no pruning. In this way, multiple trees are induced in the forest; the number of trees is pre-decided by the parameter Ntree. The number of variables (m) selected at each node is referred to as mtry or k in the literature. The depth of the tree can be controlled by a parameter nodesize (i.e. number of instances in the leaf node), which is usually set to one. Once the forest is trained or built as explained above, to classify a new instance, it is run across all the trees grown in the forest. Each tree gives classification for the new instance which is recorded as a vote. The votes from all trees are combined and the class for which maximum votes are counted (majority voting) is declared as the classification of the new instance.

IV. SIMULATION RESULTS

In this test case, we considered other standard classification scheme such as SVM, LR and RF classifier.

Parameter:

Accuracy gives a proportion of how precise your model is in anticipating the real up-sides out of the absolute up-sides anticipated by your framework. Review gives the quantity of real up-sides caught by our model by grouping these as obvious positive. F-measure can give a harmony among accuracy and review, and it is liked over precision where information is uneven.

$$Accurancy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\Pr ecision = \frac{TP}{TP + FP}$$

$$\operatorname{Re} call = \frac{TP}{TP + FN}$$

Where,

TP = True Positive,

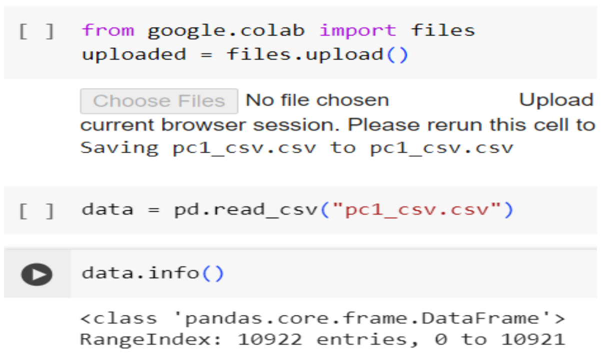
TN = True Negative

FP = False Positive,

FN = False Negative

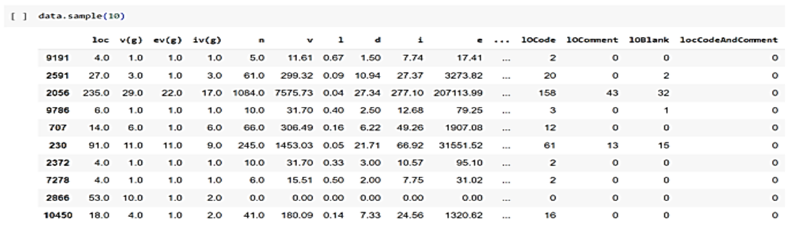
Data Information:

PC1 dataset contains total 10922 entries which are given below



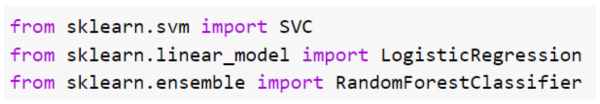
```
[ ]  from google.colab import files
     uploaded = files.upload()

     Choose Files  No file chosen                 Upload
     current browser session. Please rerun this cell to
     Saving pc1_csv.csv to pc1_csv.csv

[ ]  data = pd.read_csv("pc1_csv.csv")

     data.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 10922 entries, 0 to 10921
```

Data Sample:



```
[ ]  data.sample(10)

         loc  v(g)  ev(g)  iv(g)       n        v     l      d       i          e  ...  lOCode  lOComment  lOBlank  locCodeAndComment
9191     4.0   1.0    1.0    1.0     5.0    11.61  0.67   1.50    7.74      17.41  ...       2          0        0                  0
2591    27.0   3.0    1.0    3.0    61.0   299.32  0.09  10.94   27.37    3273.82  ...      20          0        2                  0
2056   235.0  29.0   22.0   17.0  1084.0  7575.73  0.04  27.34  277.10  207113.99  ...     158         43       32                  0
9786     6.0   1.0    1.0    1.0    10.0    31.70  0.40   2.50   12.68      79.25  ...       3          0        1                  0
707     14.0   6.0    1.0    6.0    66.0   306.49  0.16   6.22   49.26    1907.08  ...      12          0        0                  0
230     91.0  11.0   11.0    9.0   245.0  1453.03  0.05  21.71   66.92   31551.52  ...      61         13       15                  0
2372     4.0   1.0    1.0    1.0    10.0    31.70  0.33   3.00   10.57      95.10  ...       2          0        0                  0
7278     4.0   1.0    1.0    1.0     6.0    15.51  0.50   2.00    7.75      31.02  ...       2          0        0                  0
2866    53.0  10.0    1.0    2.0     0.0     0.00  0.00   0.00    0.00       0.00  ...       0          0        0                  0
10450   18.0   4.0    1.0    2.0    41.0   180.09  0.14   7.33   24.56    1320.62  ...      16          0        0                  0
```

Classifier Technique:



```python
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

Data set: in the table 1 as the predicted outcome using SVM classifier in terms of defective (D) and non-defective (ND) classes.

Table 1: PC1 Prediction for SVM

Actual class	Predicted class	
	ND	D
D	270	49
ND	15	1843

It is really tough to find a better separation between two groups. Certainly, the results obtained from the tests may overlay with one another.

Table II: PC1 Prediction for LR

Actual class	Predicted class	
	ND	D
D	327	14
ND	8	1836

Table III: Table 1: PC1 Prediction for RF

Actual class	Predicted class	
	ND	D
D	339	2
ND	0	1844

Table IV: Comparison Result

Classifier	Precision	Recall	Accuracy
SVM	99.34%	96.98%	96.84%
LR	99.56%	99.24%	98.99%
RF	100%	99.89%	99.90%

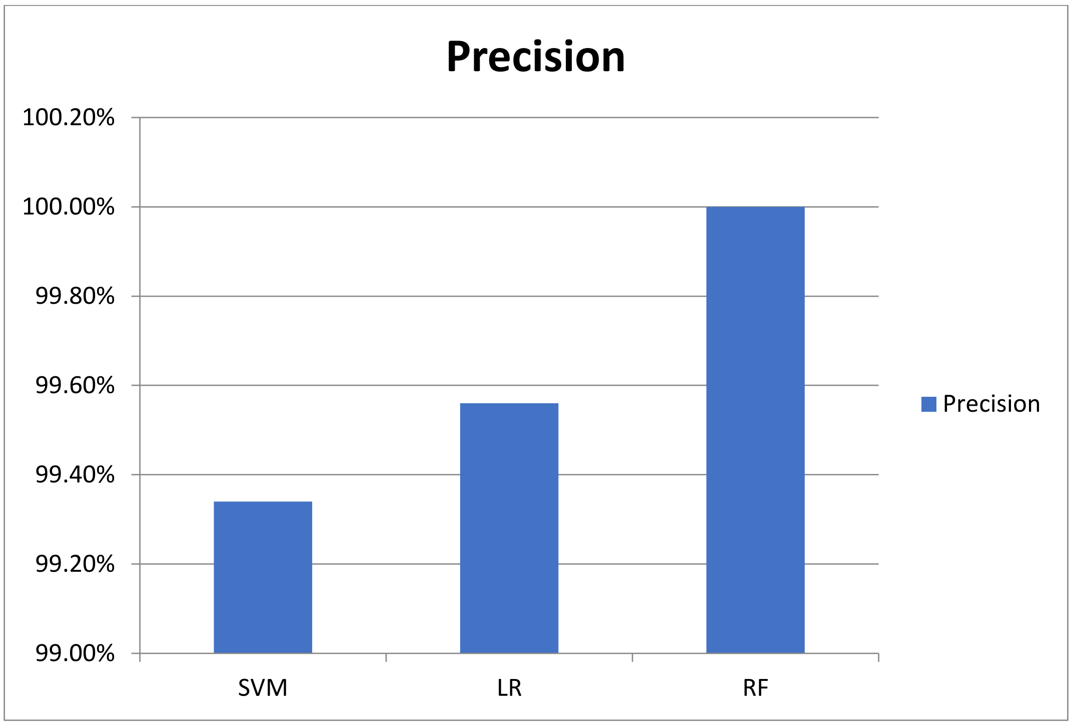


Figure 2: Graphical Represent of Precision

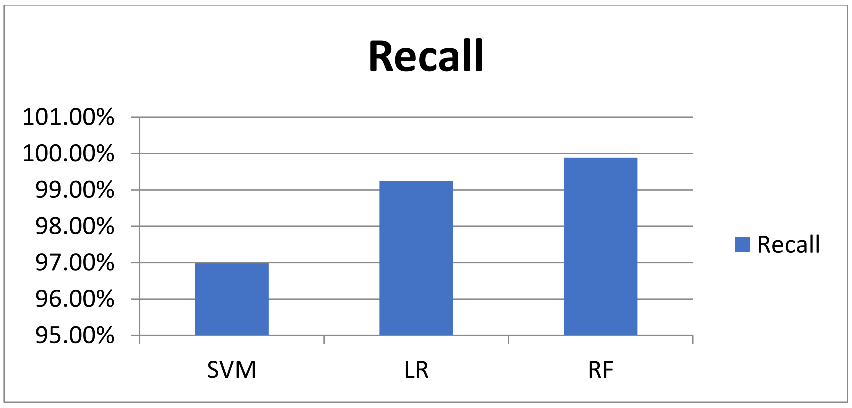


Figure 3: Graphical Represent of Recall

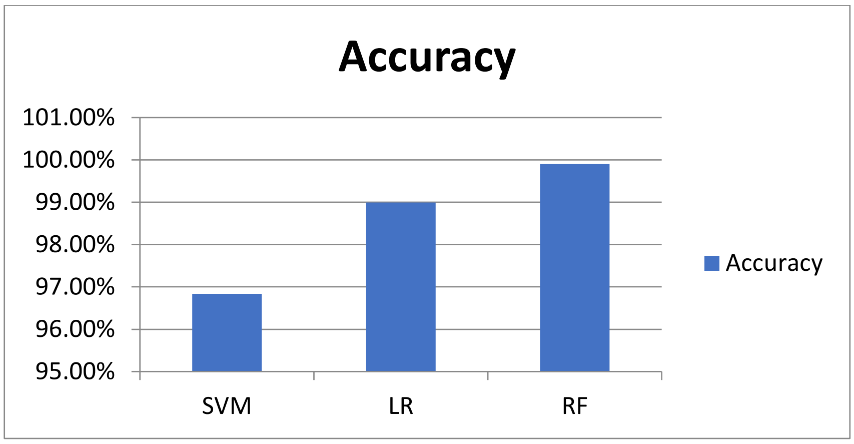


Figure 4: Graphical Represent of Accuracy

Table V displays the results of Muhammad Azam et al. [1] and implemented method in terms of accuracy. Malarvizhi et al. [1] give an accuracy of 93.05% for SVM, 93.32% for LR and 93.86% for RF. The implemented method provides an accuracy of 96.84% for SVM, 98.99% for LR and 99.90% for RF. Clearly, the implemented method is a 6.04% improvement accuracy compared to Muhammad Azam et al. [1]. Fig. 5 shows the graphical representation of the comparison result.

Table V: Comparison Results

Design	Method	Accuracy
Muhammad Azam et al. [1]	SVM	93.05%
	LR	93.32%
	RF	93.86%
Implemented Design	SVM	96.84%
	LR	98.99%
	RF	99.90%

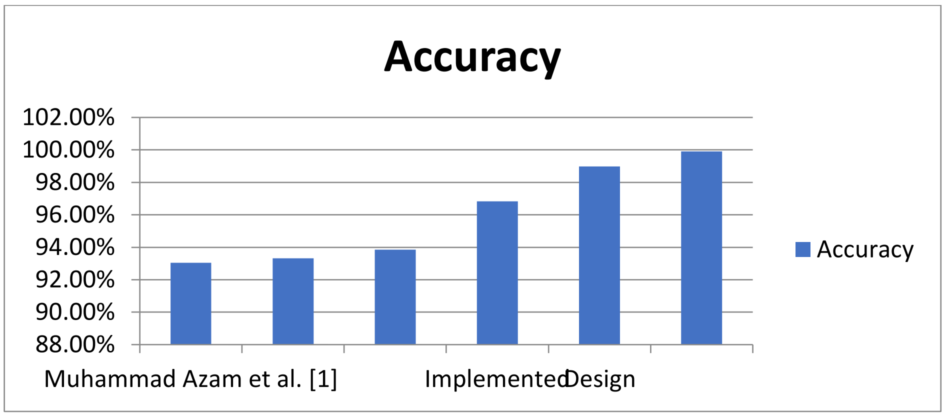


Figure 5: Graphical Represent of Accuracy

V. CONCLUSION

In this study, we investigated the application of the Random Forest machine learning algorithm for software defect prediction, with a strong emphasis on optimizing its accuracy. The results indicate that Random Forest, due to its ensemble nature and robustness against overfitting, performs exceptionally well in identifying defective software modules. By applying effective preprocessing techniques and fine-tuning the model's hyperparameters, we achieved significant improvements in prediction accuracy and overall model performance. The optimized model demonstrated strong results across multiple evaluation metrics, including precision, recall, F1-score, and AUC-ROC, proving its reliability and efficiency. This research validates the effectiveness of Random Forest as a powerful tool for software quality assurance and supports its use in real-world software development environments to proactively detect and mitigate defects. Future work may involve integrating deep learning techniques or hybrid models to further enhance defect prediction capabilities.

REFERENCES

[1] Muhammad Azam, Muhammad Nouman, Ahsan Rehman Gill, “Comparative Analysis of Machine Learning techniques to Improve Software Defect Prediction”, Journal of Computing & Information Sciences (KJCIS) Volume 5, Issue 2, pp. 41-66, 2022.

[2] Tiapride, Jaray’s, Chakri Klan Tantithamthavorn, Hao Khan Dam, and John Grundy, “An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models”, IEEE Transactions on Software Engineering 48(1):166–85, 2022.

[3] L.-Q. Chen, C. Wang, and S.-L. Song, ''Software defect prediction based on nested-stacking and heterogeneous feature selection,'' Complex Intell. Syst., vol. 8, no. 4, pp. 3333–3348, 2022

[4] M. Pavana, L. Pushpa, and A. Parkavi, ''Software fault prediction using machine learning algorithms,'' in Proc. Int. Conf. Adv. Elect. Comput. Technol., pp. 185–197, 2022.

[5] A. Al-Nusirat, F. Hanandeh, M. K. Kharabsheh, M. Al-Ayyoub, and N. Al-Dhfairi, ''Dynamic detection of software defects using supervised learning techniques", Int. J. Commun. Netw. Inf. Secur., vol. 11, no. 1, pp. 185–191, 2022.

[6] Chen, Xiang, Yinzhou Mu, Key Liu, Zhan Qi Cui, and Chao Ni, "Revisiting Heterogeneous Defect Prediction Methods: How Far Are We?", Information and Software Technology 130:106441, 2021.

[7] Esteves, Granderson, Eduardo Figueredo, Adriano Veloso, Markos Vigias, and Nivea Zaviana, "Understanding Machine Learning Software Defect Predictions", Automated Software Engineering 27(3–4):369–92, 2020.

[8] Esteves, Granderson, Eduardo Figueredo, Adriano Veloso, Markos Vigias, and Nivea Zaviana, "Understanding Machine Learning Software Defect Predictions" Automated Software Engineering 27(3–4):369–92, 2020.

[9] R. Bahaweres, F. Agustian, I. Hermadi, A. Suroso, and Y. Arkeman, ''Software defect prediction using neural network basedSMOTE,'' in Proc. 7th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI), pp. 71–76, 2020.

[10] N. Li, M. Shepperd, and Y. Guo, ''A systematic review of unsupervised learning techniques for software defect prediction,'' Inf. Softw. Technol., vol. 122, Jun. Art. no. 106287, 2020.

[11] Thota, Mahesh Kumar, Francis H. Sajin, and Gaultheria Rajesh, "Survey on Software Defect Prediction Techniques", International Journal of Applied Science and Engineering 14, 2019.

[12] Son, Le, Nakul Pritam, Manju Khari, Raghavendra Kumar, Pham Phuong, and Pham Thong, "Empirical Study of Software Defect Prediction: A Systematic Mapping", Symmetry 11(2):212, 2019.

[13] Pan, Cong, Minyan Lu, Biao Xu, and Hauling Gao, "An Improved CNN Model for Within-Project Software Defect Prediction", Applied Sciences 9(10):2138, 2019.

[14] Manjula, C., and Lilly Florence, "Deep Neural Network Based Hybrid Approach for Software Defect Prediction Using Software Metrics", Cluster Computing 22:9847–63, 2019.

[15] Al-Nusrat, Alaa, Fears Hamadeh, Mohammad Khorramshahr, Mahmoud AlAyoub, and Nahla Al-Dhahiri, "Dynamic Detection of Software Defects Using Supervised Learning Techniques", International Journal of Communication Networks and Information Security 11(1):185–91, 2019.