IRJIF. I.F. : 3.821

North Asian International Research Journal of Sciences, Engineering & I.T.

ISSN: 2454-7514 Vol.9, Issue-12 December-2023

DOI: 10.5949/nairjseit.2023.10.12.6 Page No. 23-35

SOFTWARE DEFECT PREDICTION: HANDLING IMBALANCE DATA WITH CATEGORICAL BOOSTING CLASSIFIER

[1]DR SHAIK MOHAMMAD RAFI

[1]Post-Doctoral Fellowship Scholar, Department of Computer Science and Engineering, Central Christian University, MALAWI.

[1]Professor in Computer Science and Engineering, Sri Mittapalli College of Engineering (A), Guntur &Research Supervisor, Dept. of Computer Science and Engineering.Bharath University of Research and Foundation, Tamilnadu.

Email: mdrafi.527@gmail.com

ABSTRACT

Software reliability assures that the program will operate without interruption for a certain amount of time under specific environmental conditions, whereas software quality establishes the program's worth. Software bugs affect the program's quality and dependability. The more errors there are in the program, the less reliable it is, and maintaining the product's quality takes more work. To guarantee that the final software product is more reliable and of higher quality, software quality assurance, or SQA, is an umbrella term for a variety of tasks that are used to coordinate and monitor the software development process.

In order to assist the program, achieve its intended purpose, we presented in this study an ensemble machine learning-based predictive development model for software errors. The model's performance was further evaluated using class imbalance techniques in addition to the standard evaluation metrics of accuracy, recall, f 1 measure, precision, and AUC. The Categorical Boosting Classifier exhibits an impressive classification performance of 98–100% based on the G-Mean measure of the six defect datasets utilized in this study. However, this study provides independent information that software practitioners and academics may utilize when selecting automated jobs for their intended application. The proposed model performance was compared with previous existing models, and proposed models exhibits more enhanced performance while compared with existing models.

INDEX TERMS— Defect prediction softwares; machine learning methods; SMOTE, metric softwares; prediction defect model; quality software;

I.INTRODUCTION

Nowadays, software is essential to the operation of nearly all automated and engineered systems. Because of this progress, the effectiveness of daily operations depends heavily on the dependability and quality of software systems. The software system's quality and dependability are directly impacted by how fault-prone its component parts are. Generally speaking, more errors indicate worse program quality and lower software reliability. Software is often tested in order to find and correct bugs in its component parts. A comprehensive and extensive testing is not feasible, nevertheless, because testing resources are typically scarce.

If software issues cannot be categorized in real time, they will become more difficult to find and more costly to solve. Thus, the development of automated fault prediction models is motivated by the goal of software defect prediction. If software issues are discovered prior to the program's release, the developer will have an easier time assigning and fixing them. Software defect prediction has drawn a lot of interest in the field of software-reliability engineering and is an essential part of software quality analysis [1–4].
Conversely, Menzies et al. [2] and Seiffert et al. [4] have demonstrated that class imbalance problems in real-world data sets can severely reduce the efficacy of defect predictors. When a software system is said to be "class imbalanced," it means that most of its flaws are concentrated in a very small percentage of its program modules.
Additionally, the software industry is one of the most promising sectors in the modern era and is utilizing artificial intelligence in the fourth industrial revolution as many software technologies are being automated [5]. Cutting edge machine learning techniques have been used to identify the problematic software modules from research applications and offer helpful fixes to consumers [6]. The six most popular machine learning classifiers have been utilized in this study, as advised by the most recent thorough literature analysis [7]. Each selected classification technique is tested on a range of real application datasets related to software failure prediction in the applications. The most common use case is software fault prediction using machine learning techniques, according to researchers and the software community [8].
Consequently, modern [9] machine learning algorithms have been used to the fault datasets in an effort to enhance prediction by removing superfluous information using a range of feature selection techniques and imbalance to balancing data approaches.

However, there are certain attributes that are more dependent on the quality of the data than accuracy and must be considered. A method for supervised machine learning prediction uses the predefined training data collection. The algorithm learns from the training dataset and then builds rules to predict the class label for a new collection of data. Using mathematical approaches, the prediction function is created and strengthened as part of learning. A defined output value and an attribute input value are included in the training data used by the technique. We compare the projected ML algorithm's quality with that of the well-known outcome. The training data iterations are used to continue this process until the desired prediction accuracy is reached or the maximum number of loops is reached. With respect to unsupervised learning methods, the data does not know the class label output value.
The integration of automated recovery models into software for fault recovery has been the subject of several studies [10][11][12]. Six classifiers are used to evaluate the study's purpose, which proposes an automated approach to fixing software faults. The three data sets (ANT-1.7, Camel-1.6, KC3, MC1, PC2, and PC4) from the PROMISE Software Engineering Repository [13] were utilized, along with 22

characteristics like McCabe and Halstead and a few more metrics like defect information. We pre-processed our chosen data in order to allocate relevant columns after analyzing data imbalance problems using computational methods such as SMOTE [14]. This work assesses and contrasts several machine learning techniques with the goal of forecasting software errors within software.

The next sections of the study are arranged as follows: section 2 provides a related work,section 3 contains the data sets related information, classification strategies, performance metrics, and experimental setup. Section 4 provides a description of the analysis results. Section 5 concludes with some concluding comments and illustrations for future development.

II. RELATED WORK

The capacity of a software system or its components to correct mistakes, improve performance, test systems and software, adjust to new platforms, or change development processes is referred to as software fault localization and maintainability [15].

A powerful prediction strategy for increasing the number of application software systems was offered by Wang et al. [16] using their machine learning-based software fault prediction model. Databases containing software flaws include irregular data that produces strange patterns. This problem fosters the development of a reliable and efficient contextual classifier for scientific and practical uses. The study conducted by Xu et al. [17] looked at "software defect prediction strategies and hypothesized that traditional techniques use vectorization and feature selection" framework to eliminate unimportant features while excluding other aspects that are crucial, resulting in a decrease in overall performance of the defect prediction strategy.

In order to ensure software quality, it is probable that fewer mistakes may occur throughout its operation [18]. Businesses may help to reduce the overall cost, time, and effort of software project maintenance by utilizing a software defect prediction model [19] [20].

Consequently, the classification of software module defects has a major impact on the software development process. The real problem arises, though, when an application's internal program is changed by a developer and impacts other modules—for example, making the application unresponsive to updates. As such, it's quite probable that the software may experience instability and malfunctions [21]. Felix et al. [22] presented a study that used a neural network as a machine learning technique to anticipate software faults.

Feature segment and reduction will lead to increased performance in machine learning-based classification and prediction procedures. by identifying a certain element as being essential. Lu et al. [23] used a self-study algorithm variation to construct and analyze a semi-supervised learning strategy for software fault prediction. As a solution to the uneven dataset of software defects, Jayanthi et al. [24] "established a Selection of features for apps scheme." At the conclusion of the selection process, subsets of attributes are gathered and the procedure is repeated using the attributes created on the wrapper. The following process uses random sampling to mitigate the negative impacts of the unbalanced dataset.

The method creates a robust decision area by expanding the training set's attribute ranges while preserving the same normal distribution, as proposed by Li et al. [25]. The localized generalization error model served as inspiration for Chen et al.'s ensemble learning-based approach for categorizing imbalanced data [26]. The approach proposed in reference [27] yields a limited quantity of synthetic samples that are

positioned in close proximity to the training samples. The original training samples and sampled fake neighborhoods are then combined to train the fundamental classifiers. Zhai et al. [28] proposed an oversampling method that generates positive samples inside the hyperspheres of their antagonist nearest neighbors.

Vanhoeyveld and Martens [29] empirically investigated how well-liked imbalanced learning algorithms—like oversampling—performed on sparse and large-scale behavior datasets.

III. MATERIALS AND METHODS

A. Data Collection

Random forests and W-SVMs, two basic classifiers, are used to benchmark the suggested Categorical Boosting classifier ensemble learning model. Strategies for class imbalance are also assessed. The GFS approach, Fisher's criterion, and Pearson's correlation are specifically mentioned in the selection criteria for the performance evaluation. Finally, the proposed Categorical Boosting classifier model's classification performance is evaluated using six publicly available software defect datasets:

1. Ant-1.7.
2. Camel-1.6.
3. KC3 datasets.
4. MC1.
5. PC2.
6. PC4.

Regarding the first three datasets, see [30] for further details. The last three datasets are accessible online. Despite complaints about the quality of these databases [32], scholars have made considerable use of them [31]. Table 1 contains an overview of these datasets. Notably, among other commonly used software metrics, these datasets contain LOC, McCabe's CC, branch count (BC), total number of operands (TNO), Halstead's length, volume, and difficulty.

Table 1 Defect Dataset Information

Dataset	Language	Components	Defective components
Ant-1.7	Java	745	166
Camel-1.6	Java	965	188
KC3	java	200	36
MC1	C++	1988	46
PC2	C	1585	16
PC4	C	1287	177

B. Classification Techniques

The collection includes the most widely used and well-liked machine learning techniques. The methods listed below are listed in order with brief explanations for each.

i. Naïve Bayes

Naive Bayes is a training method applied to statistical approach knowledge grouping. The word "naive" implies that this approach outright claims that the attributes of a class are unrelated. Features presume either naïve or strong isolation. It is used to construct class descriptors from finite sets and is allocated as a vector. It acts as a template that is used as class labels for problematic objects. Naive bays are categorized as challenging scenarios seen in real-world difficulties because of their assumptions and simple nature

ii. Random Forest

One popular machine learning algorithm is Random Forest, which is used in supervised learning techniques. It may be used for machine learning problems that include regression and classification. Its core concept is ensemble learning, which is the process of combining several classifiers to improve the performance of the model and solve a difficult problem.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." The random forest predicts the result based on the majority vote of predictions from each decision tree, as opposed to relying just on one.

iii. KNN (K-nearest Neighbor)

Based on the majority class label, the kNN algorithm acts as a voting mechanism when a new data point is awarded a class label among its closest 'k' (where k is an integer) neighbors in the feature space. Imagine yourself in charge of selecting a political party in a small town of a few hundred residents. To start, you may try asking your neighbors whose political party they favor. If most of your k nearest neighbors vote for party A, you'd probably vote for them too. The k nearest neighbors of a new data point in the kNN algorithm decide the class label of the data point based on the majority class label.

iv. Decision Tree

Despite being a supervised learning technique, decision trees are mostly used to address categorization problems. Regression problems can also be resolved using them, though. With core nodes representing dataset properties, branches representing decision rules, and leaf nodes representing the outcomes, this classifier is arranged like a tree.

A decision tree consists of two nodes: the Decision Node and the Leaf Node. choice nodes are used to make any form of choice and have several branches, in contrast to leaf nodes, which represent the outcome of decisions and have no more branches. The test or the judgments are based on the features of the dataset that was supplied.

v. Logistic regression

The supervised machine learning method known as logistic regression is mostly used for classification problems when attempting to predict the chance that an instance will belong to a given class or not. This kind of statistical approach looks at the relationship between a set of independent factors and the dependent binary variables. It's a useful tool for making judgments.

vi. Support Vector Machine

Support Vector Machine (SVM) is a powerful machine learning technique that can handle regression, outlier identification, and linear or nonlinear classification. Applications for Support Vector Machines (SVMs) include face recognition, anomaly detection, handwriting identification, text classification, image classification, spam detection, and gene expression analysis, among many more. SVMs work well in a variety of domains and are adaptable because they can handle high-dimensional data and nonlinear correlations. We want to find the biggest separation hyperplane—which SVM algorithms excel at finding—between the several classes that make up the target feature.

vii. Neural Network

MLPs, or multilayer perceptrons, are a type of feedforward neural network. In this artificial neural network, there are connections between each node and nodes at different levels. In his Perceptron program, Frank Rosenblatt provided the first definition of "perceptron". The fundamental unit of each artificial neuron in a neural network is called a perceptron. In this supervised learning approach, the ultimate outcome is determined by activation functions, node weights, inputs, and node values.

Multilayer perceptrons (MLPs) are exclusively used in forward-only neural networks. Every node is fully connected to the network. Every node just forwards its values to the next node in the chain.

viii. Existing Model (Issam et al [2015])[33]

By integrating feature selection with ensemble learning, the authors of this paper demonstrate how fault classification performance may be improved. A unique two-variant ensemble learning approach is proposed to provide resistance against data imbalance and feature redundancy, in addition to effective feature selection. This method may be used with or without feature selection.

A combination of efficient feature selection and well-selected ensemble learning models is employed to address these issues and decrease their influence on the defect classification performance.

According to forward selection, few attributes result in a high area under the receiver operating curve (AUC).

ix. Proposed Model(CatBoost)

Yandex created the open-source boosting library known as CatBoost, sometimes known as Categorical Boosting. It is intended for use with very large numbers of independent features in regression and classification tasks. A gradient boosting variation that can handle both numerical and categorical information is called catboost. To translate category information into numerical features, no feature encoding technology such as One-Hot Encoder or Label Encoder is needed. In order to lessen overfitting and enhance the dataset's overall performance, it also makes use of an approach known as the symmetric weighted quantile sketch (SWQS), which automatically manages the missing values in the dataset.

Binary decision trees are used as basis predictors in CatBoost, an implementation of gradient boosting. A decision tree is a model constructed by recursively dividing the feature space into many disjoint sections (tree nodes) based on the values of certain splitting characteristics. Typically, attributes are binary variables that show if a characteristic above a predetermined threshold [34].

In contrast to other gradient boosting techniques such as XGBoost and LightGBM, CatBoost creates symmetrically structured, balanced trees. This implies that the feature-split pair with the lowest loss is selected and applied to all of the nodes in that level at each phase. Many benefits come with this balanced design, including quick model application, prediction time reduction, efficient CPU implementation, and regularization to minimize overfitting. CatBoost uses the idea of ordered boosting to solve the overfitting issue on tiny or noisy data sets. Ordered boosting trains the model on one subset of data while computing residuals on another, in contrast to traditional boosting methods that utilize the same data instances for gradient estimation as the ones used to train the model. This method aids in avoiding overfitting and target leakage.

[35] One such method is the Synthetic Minority Oversampling technique (SMOTE). SMOTE is used to handle imbalanced datasets by synthesizing fake samples for the minority class. In order to solve class imbalance, this research looks at the significance of SMOTE and how it may be used to improve classifier model performance. By lowering bias and incorporating important minority class traits, SMOTE enhances model performance and increases the accuracy of outcome predictions.

C. Performance Measurement

Once the predictive model is built, it may be tested to predict the fault modules in the software defect dataset. We evaluated the ML prediction models in this work using six classification techniques based on different statistical methodologies [36], such as confusion matrix (True Positive = TP, True Negative = TN, False Positive = FP, False Negative = FN), recall, precision, F1 measure, etc. Table 3 shows the quality metric of a predictive model based on a confusion matrix as follows [37].

The area under the curve, or AUC curve, is a representation of the region beneath the ROC curve. It assesses the general performance of the binary classification model. Since both TPR and FPR range from 0 to 1, the region will always fall between those two numbers. Improved model performance is shown by a higher AUC value. Our main goal is to enhance this area in order to have the best TPR and lowest FPR at the designated threshold. The chance that a randomly chosen positive event would be given a higher projected probability by the model than a randomly chosen negative instance is shown by the area under the curve (AUC).

Table 2 Performance Measurement Criteria

Metrics	Mathematical expression
Accuracy	$\frac{(TP+TN)}{(TP+FP+TN+FN)}$
Precision	$\frac{TP}{(TP+FP)}$
Recall	$\frac{TP}{(TP+FN)}$
F1 score	$\frac{2*(Recall*Precision)}{(Recall+Precision)}$
Specificity	$\frac{TN}{(TN+FP)}$
G-Mean	$\sqrt{Sensitivity*Specificity}$

The geometric mean (G-mean) is a commonly used performance measure in classification problems with

imbalanced datasets [38]. This measure displays the classifier's performance on both the majority and minority classes. This evaluation makes use of specificity and sensitivity metrics.

Table III. Classification results of proposed model versus other classification models (AUC measure).

Classification Algorithms	KNN	Decision Tree	Logistic Regression	SVM	Random forest	Navie Bayes	MLP	Issam H. Laradji Model	CatBoost
Ant-1.7	0.82	0.80	0.74	0.70	0.89	0.67	0.79	0.86	0.882
Camel-1.6	0.74	0.76	0.65	0.60	0.89	0.58	0.68	0.80	0.859
KC3	0.61	0.84	0.66	0.64	0.89	0.64	0.61	0.86	0.875
MC1	0.96	0.99	0.88	0.78	1.00	0.79	0.60	0.98	0.994
PC2	0.95	0.99	0.87	0.71	1.00	0.66	0.92	0.95	0.992
PC4	0.76	0.92	0.75	0.59	0.96	0.66	0.59	0.96	0.961

Table IV. Classification results of proposed model versus other classification models (G-mean measure).

Ant-1.7	**0.82**	**0.79**	**0.74**	**0.69**	**0.88**	**0.65**	**0.78**	**0.84**	**0.88**
Camel-1.6	0.72	0.76	0.65	0.55	0.89	0.48	0.67	0.79	0.86
KC3	0.60	0.84	0.65	0.63	0.89	0.60	0.47	0.83	0.87
MC1	0.96	0.99	0.88	0.76	1.00	0.77	0.46	0.90	0.99
PC2	0.95	0.99	0.86	0.67	1.00	0.59	0.92	0.95	0.99
PC4	0.75	0.92	0.75	0.54	0.96	0.58	0.49	0.79	0.96

D. Experimental Setup

The methodology of the experiment and the recommended software fault predictive development paradigm are described in this section. Predicting software issues is an essential duty in the field of software engineering. The preceding chapter provides an explanation of machine learning software-focused fault prediction techniques. Additionally, these methods addressed software bug mismatch problems, although overall efficiency and classification accuracy remain challenges for academics.

To solve this issue, a class imbalance approach called SMOTE scheme is developed in conjunction with an excessively Categorical Boosting classification technique for software fault prediction. The overall process of defect prediction is depicted in Fig. 1. In the initial stage of dataset preprocessing, techniques were employed to identify any abnormalities or missing data.

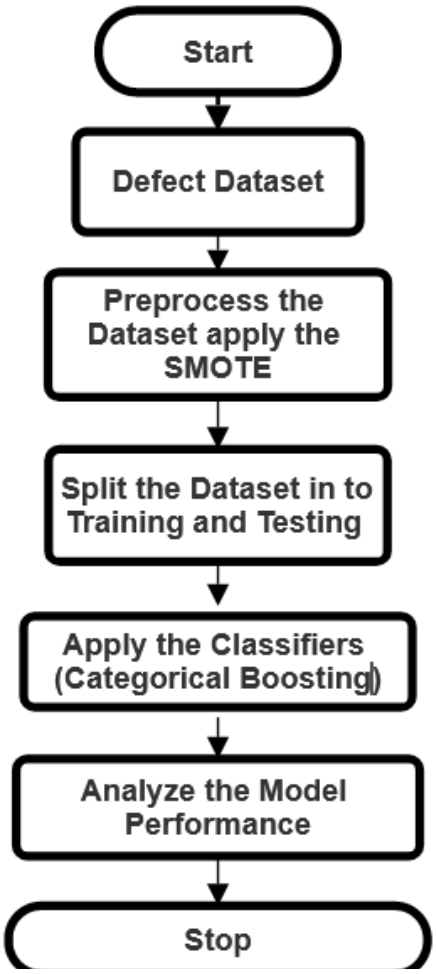
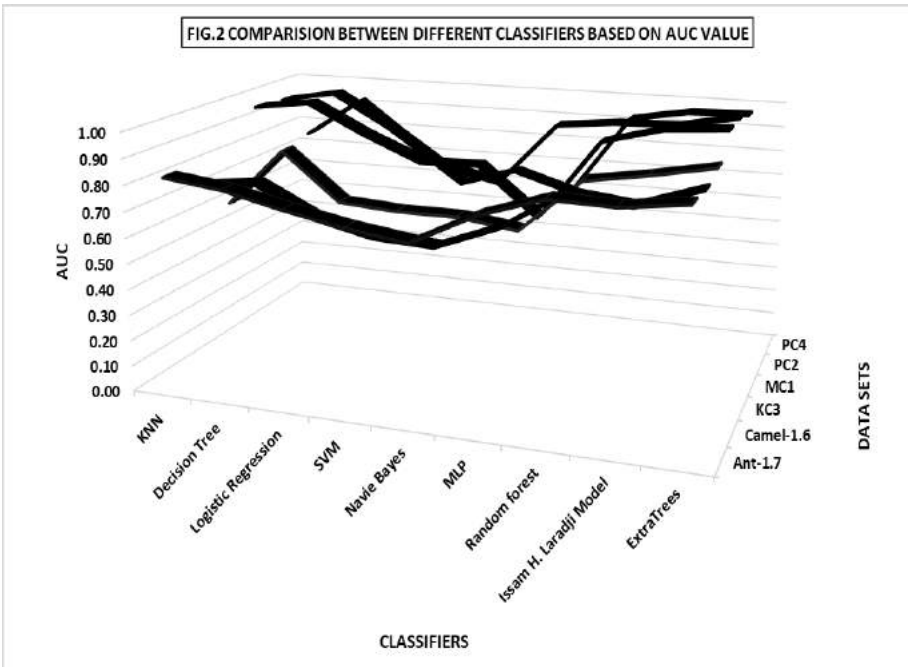
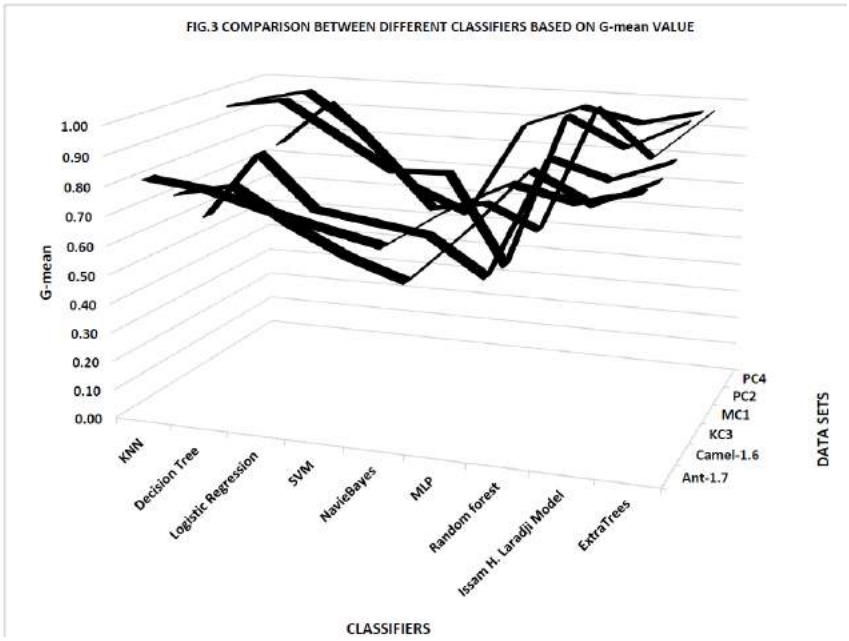


Fig 1. Work flow of the Defect model





IV. RESULTS AND DISCUSSION

In our research, we noticed both defective and non-faulty classes; nevertheless, the automatic fault recovery inside software through the use of a predictive model was the major focus of our attention. Faulty modules are almost always quite critical. We used the SMOTE class imbalance approach in our experiment with severely Categorical Boostingclassification to evaluate and compare the performance of various classification techniques. To determine the parameters for the software defect model, a number of data pretreatment techniques that have increased the consistency and accuracy of the classification model were used.

Software failure prediction approaches' performance evaluation is displayed in Tables 2 and 3. Regarding the AUC

and G-mean, our suggested model completed the best (i.e., 100%) on ANT-1.7, Camel-1.6, KC3, MC1, PC2 and PC4 datasets. Fig 2 and 3 shows the performance of each classifier with respect to different datasets. On the whole the performance of Categorical Boosting trees shows the enhanced performance with respect to AUC and G-mean metric value.

V. CONCLUSION

Estimating software defects by information-mining techniques is the main goal of this research. In this field, which is now a hot topic for research, several techniques have also been looked at to increase the efficiency of finding bugs or software issues. This work built a unique hybrid model that combines classification and class balancing approach to solve the issue of classification accuracy for huge datasets. SMOTE retrieves data, which is then used for a further classification step, yielding a class balance model. Categorical Boosting tree classification method is also used for detecting program mistakes.

There is no appreciable change in the classifier accuracy whether the class imbalance technique is not utilized or when the class imbalance approaches are applied. The accuracy of the classifier varies depending on whether a collection of characteristics is present or whether class-balanced procedures are being applied. It follows that using class balance techniques lowers bias and variance for fault prediction without lowering prediction accuracy. More improvements to these results can be achieved by using multiple datasets. With more datasets, the outcomes could be better. Another alternative is to compare additional ways.

The most widely used and well-liked strategies were considered in this study. New methods are expected to be demonstrated in the future and used to the thorough analysis.

REFERENCES

[1]T. M. Khoshgoftaar, E. B. Allen, and J. Deng, “Using Regression Trees to Classify Fault-Prone Software Modules,” IEEE TRANSACTIONS ON RELIABILITY, vol. 51, no. 4, 2002.

[2] 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.

[3] Y. Ma, G. Luo, X. Zeng, and A. Chen, “Transfer learning for cross-company software defect prediction,” Information and Software Technology, vol. 54, no. 3, pp. 248–256, 2012.

[4] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, “Improving software-quality predictions with data sampling and boosting,” IEEE Transactions on Systems, Man, and Cybernetics A, vol. 39, no. 6, pp. 1283–1294, 2009.

[5] H. B. Bolat, G. T. Temur, and IGI Global, Agile approaches for successfully managing and executing projects in the fourth industrial revolution.

[6] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” Appl. Soft Computing. vol. 27, pp. 504–518, Feb. 2015.

[7] L. Son et al., “Empirical Study of Software Defect Prediction: A Systematic Mapping,” Symmetry (Basel)., vol. 11, no. 2, p. 212, Feb. 2019.

[8] G. P. Bhandari and R. Gupta, “Machine learning based software fault prediction utilizing source code metrics,” in 2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS), 2018, pp. 40–45.

[9] C. W. Yohannese and T. Li, "A Combined-Learning Based Framework for Improved Software Fault Prediction," Int. J. Comput.Intell. Syst., vol. 10, no. 1, p. 647, Dec. 2017.

[10] A. Hudaib et al., "ADTEM-Architecture Design Testability Evaluation Model to Assess Software Architecture Based on Testability Metrics," J. Softw. Eng. Appl., vol. 08, no. 04, pp. 201–210, Apr. 2015.

[11] S. Elmidaoui, L. Cheikhi, and A. Idri, "Towards a Taxonomy of Software Maintainability Predictors," Springer, Cham, 2019, pp. 823–832.

[12] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 367–377.

[13] "PROMISE DATASETS PAGE." [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasetspage. html. [Accessed: 01-Jul-2019].

[14] R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," Neurocomputing, vol. 343, pp. 120–140, May 2019.

[15] 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology.

[16] Wang, T., Zhang, Z., Jing, X., Zhang, L.: Multiple kernel ensemble learning for software defect prediction. Autom.Softw.Eng. 23, 569–590 (2015).

[17] Xu, Z., Xuan, J., Liu, J., Cui, X.: MICHAC: defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suita, pp. 370–381 (2016).

[18] I. U. Nisa and S. N. Ahsan, "Fault prediction model for software using soft computing techniques," in 2015 International Conference on Open Source Systems & Technologies (ICOSST), 2015, pp. 78–83.

[19] P. Oman and J. Hagemeister, "Construction and testing of polynomials predicting software maintainability," J. Syst. Softw., vol. 24, no. 3, pp. 251–266, Mar. 1994.

[20] T. Anderson, P. A. Barrett, D. N. Halliwell, and M. R. Moulding, "Software Fault Tolerance: An Evaluation," IEEE Trans. Softw. Eng., vol. SE-11, no. 12, pp. 1502–1510, Dec. 1985.

[21] S. N. Ahsan and F. Wotawa, "Fault Prediction Capability of Program File's Logical-Coupling Metrics," in 2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement, 2011, pp. 257– 262.

[22] Felix, E.A. and Lee, S.P., 2017. Integrated approach to software defect prediction. IEEE Access, 5, pp.21524-21547.

[23] H. Lu, B. Cukic, and M. Culp, "Software defect prediction using semisupervised learning with dimension reduction," in Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on. IEEE, 2012, pp. 314–317.

[24] Jayanthi, R. and Florence, L., 2019. Software defect prediction techniques using metrics based on neural network classifiers. Cluster Computing, 22(1), pp.77-88.

[25] Q. Li, G. Li, W. J. Niu, et al. Boosting imbalanced data learning with Wiener process oversampling. Frontiers of Computer Science, 2017, 11(5):836-851.

[26] X. Z. Wang, Q. Y. Shao, Q. Miao, et al. Architecture selection for networks trained with extreme learning machine using localized generalization error model. Neurocomputing, 2013,102:3-9.

[27] Z. Chen, T. Lin, X. Xia, et al. A synthetic neighborhood generation based ensemble learning for the imbalanced data classification. Applied Intelligence, 2017, https://doi.org/10.1007/s10489-017-1088-8.

[28] J. H Zhai, S. F. Zhang, M. Y. Zhang, et al. Fuzzy integral-based ELM ensemble for imbalanced big data

classification. Soft Computing, 22(11):3519-3531.
[29] J. Vanhoeyveld, D. Martens. Imbalanced classification in sparse and large behavior datasets. Data Mining and Knowledge Discovery, 2018, 32(1):25-82.
[30] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, B. Turhan, The PROMISE repository of empirical software engineering data, 2012.
[31] D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson, The misuse of the NASA metrics data program data sets for automated software defect prediction, in: Evaluation & Assessment in Software Engineering EASE 25, 2011, pp. 12–25.
[32] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Trans. Softw. Eng. 38 (2012) 1276–1304.
[33] Issam H. Laradji, Mohammad Alshayeb , LahouariGhouti "Software defect prediction using ensemble learning on selected Features" Information and Software Technology 58 (2015) 388–402
[34] LiudmilaProkhorenkova, GlebGusev, AleksandrVorobev, Anna VeronikaDorogush, AndreyGulin "CatBoost: unbiased boosting with categorical features" 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada
[35] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer "SMOTE: Synthetic Minority Over-sampling Technique" Journal Of Artificial Intelligence Research, Volume 16, pages 321-357, 2002
[36] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-oriented software fault prediction using neural networks," Inf. Softw. Technol., vol. 49, no. 5, pp. 483–492, May 2007.
[37] P. N. Tan, M. Steinbach, and V. Kumar, Introduction to data mining. Pearson Addison Wesley, 2005
[38] G. Nguyen, A. Bouzerdoum, S. Phung, Learning pattern classification tasks with imbalanced data sets, Pattern Recogn. (2009) 193–208.

Authors Information:-

Dr Shaik Mohammad Rafi is currently a Professor& Head in the faculty of Computer Science and Artificial Intelligence at the Sri Mittapalli College of Engineering, Affiliated to Jawaharlal Nehru Technological University Kakinda, India. Received his undergraduate degrees (B-Tech in Computer Science and Engineering) from the Jawaharlal Nehru Technological University Hyderabad, Telengana, India as well as his M-Tech (Computer Science and Engineering) Master's degree from AcharyaNagarjuna State University – Guntur, Andhra Pradesh, India, and his PhD in Computer Science and Engineering from AcharyaNagarjuna State University – Guntur, Andhra Pradesh, India.Published number of papers in preferred Journals and chapters in books. Life member of CSI and ISTE and member of IEEE, Presented various academic as well as research-based papers at several national and international conferences. His research activities are currently twofold: while the first research activity is set to explore the design and development of various machine and deep learning algorithms; the second major research theme that he is pursuing is focused on the applications of these machine learning and deep learning algorithms in various real time computer science applications.