Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 2458-9403
Vol. 12 Issue 5, May - 2025

COMPARATIVE PERFORMANCE ASSESSMENT OF LINEAR SUPPORT VECTOR MACHINE AND RANDOM FOREST MODEL FOR SOFTWARE DEFECT PREDICTION

Nwachukwu-Nwokeafor Kenneth C[1]
Department of Computer Engineering,

Michael Okpara University of Agric, Umudike,
Nwachukwu.nkenneth @mouau.edu.ng, nwachukwuken72@gmail.com

Simeon Ozuomba[2]
Department of Computer Engineering,
University of Uyo, Akwa Ibom, Nigeria
simeonoz@yahoomail.com , simeonozuomba@uniuyo.edu.ng

Philip Asuquo[3]

Department of Computer Engineering,
University of Uyo, Akwa Ibom, Nigeria
philipasuquo@uniuyo.edu.ng

Stephen Bliss U.[4]

Department of Computer Engineering,
University of Uyo, Akwa Ibom, Nigeria
blissstephen@uniuyo.edu.ng

Abstract*— generally, software defect datasets are highly imbalanced thereby affecting the bug prediction model performance. Therefore, in this work, comparative performance assessment of Linear Support Vector Machine (LinearSVC) and Random Forest (RF) model for software defect prediction is presented. Specifically, the essence of this study is to evaluate the two machine learning models' performance when trained with imbalanced labelled software defect datasets and also when trained with the balanced labelled software defect datasets. Labelled software defect datasets obtained from NASA Metrics Data Program (MDP) repository was used for the study and the Synthetic Minority Over-sampling Technique (SMOTE) was used for class balancing of the minority class. The results show that the LinearSVC has higher accuracy of 91.45 % in the baseline case without data balancing while the Random Forest outperformed the LinearSVC in the SMOTE data balanced case with accuracy of 92.86%. Although, the LinearSVC performed better with imbalanced dataset, however, its accuracy dropped by about 3.59% when the dataset is balanced, whereas, the Random Forest performed better with balanced dataset as its accuracy increased by about 2,73% when the dataset is balanced using SMOTE. Furthermore, precision, recall, F1 score, AUC and training time of the LinearSVC and the Random Forest increased when the balanced dataset is used. Essentially, data balancing can generally improve the model performance while at the same time increase the training time. Finally, the results showed that the LinearSVC is more suitable for imbalanced dataset while the random Forest is the best model in the case of balanced dataset.

Keywords*— Linear Support Vector Machine, Synthetic Minority Over-Sampling Technique (SMOTE), Random Forest Model, Data Balancing, Software Defect Prediction

1. Introduction

Nowadays, machine learning is increasing used to enhance the software development process [1,2]. Particularly, in the area of software defect prediction, researchers have gone a long way to train and evaluate several models that can efficiently predict the presence of defects in software artifacts [3,4]. The use of such machine learning tools can lead to autonomous software defect detection and correction mechanisms [5,6]. However, one of the major challenges of the software defect prediction models is that software defect datasets are highly imbalanced [7,8]. This requires careful balance among the key performance parameters such as, accuracy and training time.

Accordingly, this study, focuses on evaluating two machine learning models performance using the imbalanced dataset as the baseline and the balanced dataset as the target case. Particularly, in this study the Linear Support Vector Machine (Linear SVC) and Random Forest (RF) model are used for the Software Defect Prediction (SDP) on a collection of labelled software defect datasets [9,10]. The Synthetic Minority Over-sampling Technique (SMOTE) was used for the class balancing of the dataset [11,12]. The performance of the defect prediction models in terms of accuracy, precision, recall, F1 score, AUC and training time are evaluated for the baseline case with imbalanced dataset and for the target case with SMOTE balanced dataset. The essence of the study is to identify which model is suitable for the imbalanced dataset case and for the balanced dataset case. It is also the focus of the

study to identify how the data balancing affect each of the key performance parameters of the prediction models.

2. Methodology

The essence of this study is to evaluate two different machine learning models' performance when trained with imbalanced labelled software defect datasets and also when trained with the balanced labelled software defect datasets. Particularly, in this study the Linear Support Vector Machine (Linear SVC) and Random Forest (RF) model are used for Software Defect Prediction (SDP) on a collection of labelled software defect datasets obtained from NASA Metrics Data Program (MDP) repository. The datasets have diverse range of metrics including size, complexity (e.g., McCabe metrics), and Halstead metrics. Also, the datasets have imbalanced class distribution, which is common in defect prediction datasets due to the naturally lower occurrence of defective modules compared to non-defective ones. After the datasets are pre-processed, the data was split into 70/30 for training and validation respectively. Then, the Random Forest (RF) and the Linear SVC models were trained with the imbalanced datasets. This is referred to as the baseline case.

Furthermore, the Synthetic Minority Over-sampling Technique (SMOTE) was used for class balancing of the dataset. Again, the balanced datasets were split into 70/30 for training and validation respectively. Then, the Random Forest (RF) and the Linear SVC models were again trained with the balanced datasets. The models performance are compared in terms of accuracy, precision, recall, F1 score and training time.

The procedure used for the RF classification model is presented as Algorithm 1. The procedure used for the Linear SVC model is presented as Algorithm 2 while the procedure for the SMOTE data balancing is presented as Algorithm 3.

Algorithm 1: The Procedure for the Random Forest Model

Step 1: Begin
Step 2: Inputs:
Step 3: Training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$
Step 4: Number of decision trees B
Step 5: Total number of features d
Step 6: Number of features to consider at each split $m = \sqrt{d}$
Step 7: Output:
Step 8: Ensemble classifier $H(x) = mode(h_1(x), h_2(x), \dots, h_B(x))$
Step 9: Initialize Random Forest
Step 10: Set B the number of trees to be grown
Step 11: For each tree $b = 1$ to B:
Step 12: Draw a bootstrap sample D_b of size n from D with replacement
Step 13: Train decision tree h_b
Step 14: For each node of the tree:
Step 15: Randomly set $m \leq d$ features $\mathcal{F} \subset \{1, 2, \dots, d\}$
Step 16: Find the best split over $\mathcal{F}$ that maximizes the impurity (Gini Index)
Step 17: If max_depth is attained, elfseif the minimum number of samples per node is satisfied)
Step 18: Store the trained tree $h_b(x)$
Step 19: End for
Step 20: Finalize the ensemble model $H(x) = mode(h_1(x), h_2(x), \dots, h_B(x))$
Step 21: End for
Step 22: Predict Bug
Step 23: Given a test input $x \in R^d$
Step 24: For each tree $h_b \in H$
Step 25: compute prediction $y_b = h_b(x)$
Step 26: Aggregate prediction through majority vote: $\hat{y} = \arg\max_{k \in \{0,1\}} \sum_{b=1}^{B} (h_b(x) = k)$
Step 27: compute OOB prediction: $\hat{y}_{OOB}^{(i)} = mode(\{h_b(x^{(i)}) \mid b \in B_i\})$
Step 28: return predicted values

Step 29: end for
Step 30: end

Algorithm 2: The Procedure for the Linear SVC Model

Step 1: Begin
Step 2: Input:
Step 3: Dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$
Step 4: Regularization parameter C
Step 5: Tolerance ϵ
Step 6: Maximum iteration T_{max}
Step 7: Output:
Step 8: weight vector w, Intercept b
Step 9: Set $w^{(0)} = 0$, $b^{(0)} = 0$
Step 10: Initialize dual variables $\alpha^{(0)} = 0$
Step 11: if $dual\ is\ True$:
Step 12: solve the dual problem using coordinate descent:
Step 13: $\min_{\alpha} \left(\frac{1}{2}\right)(\alpha^T)Q\alpha - (e^T)\alpha$, subject to $0 \leq \alpha_i \leq C$, $i = 1,2,\ldots,n$

Where, $Q_{ij} = \left((y_i)(y_j)(x_i^T)(x_j)\right) + \delta_{ij}$

$e = 1$ (that is, vector of ones)

Step 14: end if
Step 15: for training sample or coordinate i where $i\ is\ a\ member\ of\ \alpha$
Step 16: Compute gradient: $G_i = y_i w^T x_i - 1$
Step 17: Compute: $PG_i = \begin{cases} \min(0, G_i) & if\ \alpha_i = 0 \\ \max(0, G_i) & if\ \alpha_i = C \\ G_i & Otherwise \end{cases}$ where PG is the projected gradient
Step 18 if $|PG_i| < \epsilon$
Step 19: Skip update for i
Step 20: else
Step 21: Update: $\alpha_i \leftarrow \alpha_i - \frac{G_i}{Q_{ii}}$, where Q_{ii} is the dual Hessian metric for the i instance
Step 22: $\alpha_i \leftarrow min\left(C, max(0, \alpha_i)\right)$ // (project this value into box constraint)
Step 23: Update weight vector: $w \leftarrow w + (\alpha_i^{t+1} - \alpha_i^t) y_i x_i$
Step 24: end if
Step 25: if convergence is reached
Step 26: compute intercept: $b = \frac{1}{|S|}\sum_{i \in S}(y_i - w^T x_i)$

Where $S = \{i \mid 0 \leq \alpha_i \leq C\}$ is the set of support vectors
Step 27: end if
Step 28: end for
Step 29: For a given input x, predict the class label as: $\hat{y} = sign(w^T x + b)$
Step 30: end

Algorithm 3: The Procedure for the Synthetic Minority Over-sampling (SMOTE)

Step 1: Input:

(ii) Dataset: $D = \{x_i, y_i\}_{i=1}^{n}$
(iii) Minority class samples: $X_{minority} = \{x_1, x_2, \ldots, x_n\}$
(iv) Number of synthetic samples to generate: N
(v) Number of nearest neighbours: k
(vi) New samples control factor: λ

Step 2: Output: Augmented minority class set with synthetic samples: $X'_{minority}$

Step 3: For each x_i in $X_{minority}$:

Step 4: Find the k nearest neighbours of x_i within $X_{minority}$ using Euclidean distance

Step 5: For every one of the synthetic sample which will be created:

Step 6: Randomly choose one of the k nearest neighbours, denoted x_j.

Step 7: Create x_{new} using $x_{new} = x_i + \lambda \cdot \left(x_i^{(k)} - x_i\right)$

Step 8: Append x_{new} to $X'_{minority}$.

Step 9: end for

Step 10: return $X'_{minority}$ as the new balanced minority dataset.

Step 11: end for

Step 12: end

3. Results and Discussion

The summary of the NASA-MDP datasets is presented in Table 1. The dataset has about 13 different files each different software historical data records, as Shown in Table 1. The screenshots in Figure 1 and Figure 2 show the first-five rows (original data) and the last-five rows (o SMOTE generated synthetic) data for each of the CM1.arff dataset. The summary of the statistical description of the synthetic and original CM1.arff dataset at 95% confidence level is presented in Table 2.

Table 1: The summary of the NASA-MDP datasets is presented in

File Name	Attribute	Total Instances	Defected (1)	Non-Defected (0)	Programming Language
CM1.arff	Defective	344	42	302	C
JM1.arff	label	9593	1759	7834	C
KC1.arff	Defective	2096	325	1771	C++
KC3.arff	Defective	200	36	164	C
KC4.arff	Defective	0	0	0	Not Specified
MC1.arff	Defective	9277	68	9209	C++
MC2.arff	Defective	127	44	83	C
MW1.arff	Defective	264	27	237	C
PC1.arff	Defective	759	61	698	C
PC2.arff	Defective	1585	16	1569	C++
PC3.arff	Defective	1125	140	985	Java
PC4.arff	Defective	1399	178	1221	C++
PC5.arff	Defective	17001	503	16498	C#

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	...	PERCENT_COMMENTS	LOC_TOTAL	Defective
0	6.0	9.0	2.0	...	4.00	25.0	0
1	15.0	7.0	3.0	...	39.22	32.0	1
2	27.0	9.0	1.0	...	47.27	33.0	1
3	7.0	3.0	2.0	...	0.00	12.0	0
4	51.0	25.0	13.0	...	11.67	106.0	0

[5 rows x 38 columns]

Figure 1: The screenshot showing the first-five rows of the original CM1.arff dataset

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	...	PERCENT_COMMENTS	LOC_TOTAL	Defective
599	31.051909	29.685800	16.896181	...	43.158550	92.579236	1
600	12.087939	7.182412	3.000000	...	43.132352	36.824122	1
601	16.661811	13.089275	3.516739	...	28.194191	65.178550	1
602	6.172676	15.230235	5.654647	...	18.487691	63.057559	1
603	1.801884	4.099058	2.000000	...	32.785848	13.198116	1

[5 rows x 38 columns]

Figure 2: The screenshot showing the last-five rows of the synthetic c dataset based on SMOTE

Table 2**: Random Forest model performance**

	Accuracy	**Precision**	**Recall**	**F1 Score**	**AUC**	**Training Time (s)**
Baseline **Random Forest**	0.901316	0.285714	0.166667	0.210526	0.849702	0.347974
SMOTE **Random Forest**	0.928571	0.894737	0.971429	0.931507	0.986097	0.699759

Table 3**: LinearSVC model performance**

	Accuracy	**Precision**	**Recall**	**F1 Score**	**AUC**	**Training Time (s)**
Baseline Linear SVC	0.914474	0.400000	0.166667	0.235294	0.682738	0.138527
SMOTE Linear SVC	0.878571	0.819277	0.971429	0.888889	0.923622	0.383124

The model performance for Random Forest is presented in Table 2 while Table 3 presents the LinearSVC model performance. The bar chart for the comparison of the accuracy of the Random Forest and the LinearSVC models is presented in Figures 3. The bar chart for the comparison of the normalized training time of the Random Forest and the LinearSVC models is presented in Figures 4.The bar chart for the percentage change in the performance parameters of the Random Forest and the LinearSVC models is presented in Figures 5.

According to the results in Table 2 and Table 3, as well as Figure 3, the LinearSVC has higher accuracy of 91.45 % in the baseline case without data balancing. However, the Random Forest outperformed the LinearSVC in the SMOTE data balanced case with accuracy of 92.86%. The results showed that the LinearSVC performed better with imbalanced dataset while its accuracy dropped by about 3.59% when the dataset is balanced, as shown in Figure 3 and Figure 5. On the other hand, the Random Forest performed better with balanced dataset as its accuracy increased by about 2,73% when the dataset is balanced, as shown in Figure 3 and Figure 5.

Also, the results in Table 3 and Table 4, as well as Figure 4, show that the Random Forest requires larger training time than the LinearSVC. Also, the training time for the balanced dataset is higher than that of the imbalanced dataset. This can be explained is due to the fact that data balancing introduced extra data items thereby increasing the number of data items to be processed in the balanced dataset. Remarkably, the results in Figure 5 show

it is only accuracy of the LinearSVC that dropped while all the other performance parameters increased in their values for the balanced dataset: the precision, recall, F1 score, AUC and training time of the LinearSVC and the Random Forest increased when the balanced dataset is used. Essentially, data balancing can generally improve the model performance while at the same time increase the training time. However, where the data balancing entails downsizing of the dataset, the training time may as well reduce with the downsizing. In any case, the results show that the Random forest model is preferred in the case where data balancing is applied whereas the LinearSVC is more suitable for imbalanced dataset as it has both higher accuracy and lower training time in such case. In general, the result showed that data balancing has positive impact in the performance parameters of the models and the training time can be maintained or minimized by appropriate choice of data balancing approach; whether to balance by adding more data items, dropping some data items or maintaining the total number of data items while adjusting the proportions of each data class in the balanced dataset.

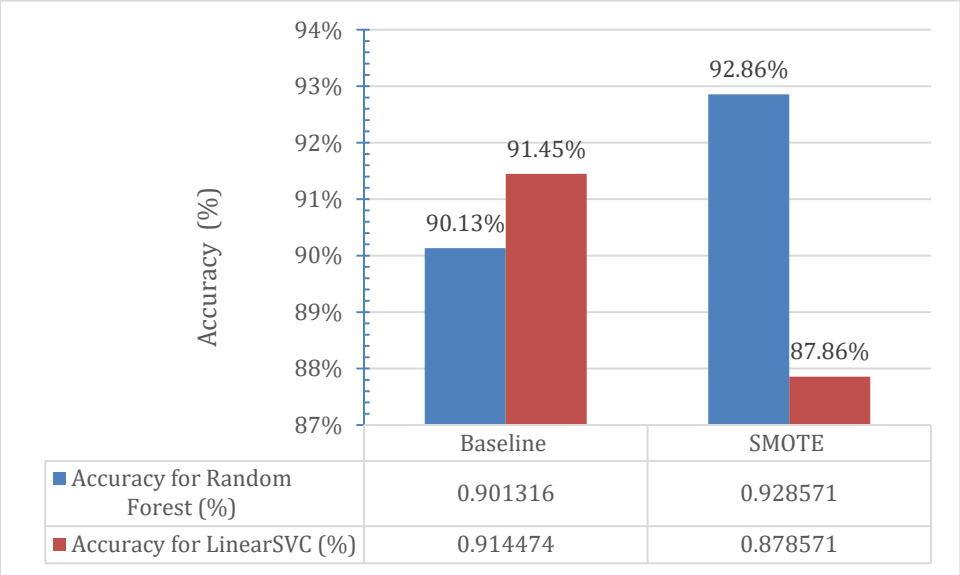


Figure 3: The bar chart for the comparison of the accuracy of the Random Forest and the LinearSVC models

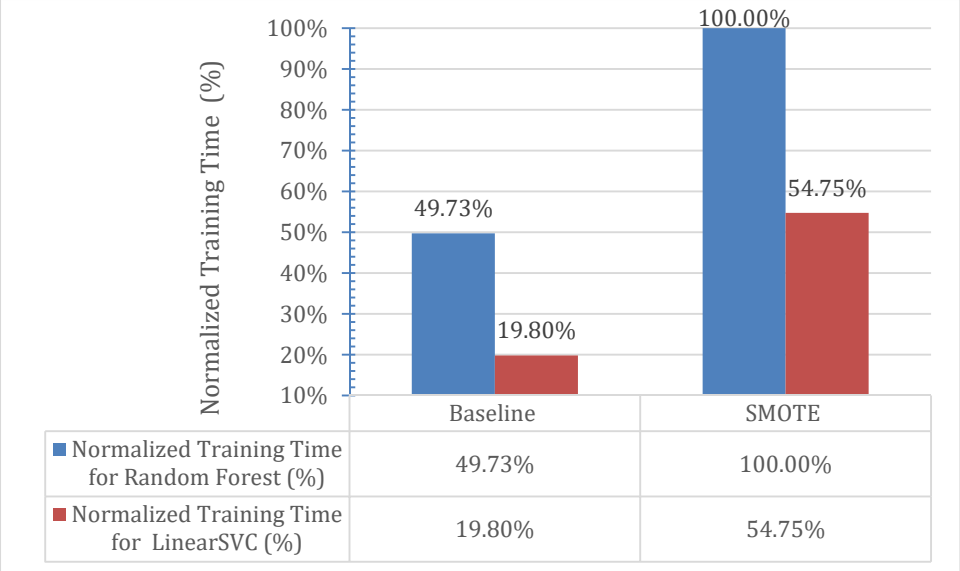
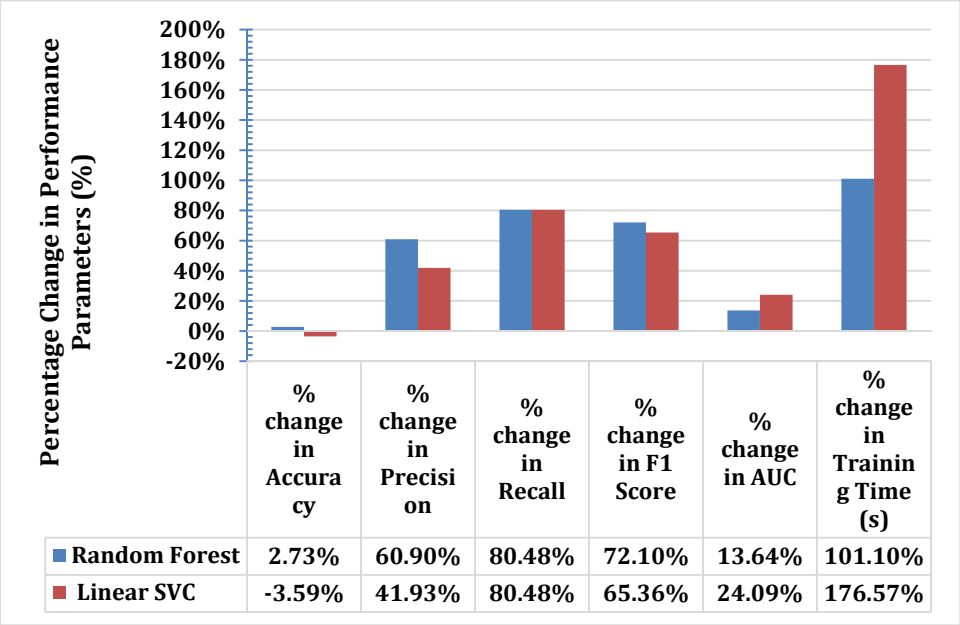


Figure 4: The bar chart for the comparison of the normalized training time of the Random Forest and the LinearSVC models



	% change in Accuracy	% change in Precision	% change in Recall	% change in F1 Score	% change in AUC	% change in Training Time (s)
Random Forest	2.73%	60.90%	80.48%	72.10%	13.64%	101.10%
Linear SVC	-3.59%	41.93%	80.48%	65.36%	24.09%	176.57%

Figure 5: The bar chart for the percentage change in performance parameters of the Random Forest and the LinearSVC models

4. Conclusion

The study presents random Forest and Linear Support Vector Machine (Linear SVC) models for the prediction of software bugs. The software defect datasets are generally highly imbalanced. Hence, the emphasis in this study is on the evaluation of the impact of data balancing on the performance of each of the two models. The two models were trained and evaluated in the baseline case without data balancing. Afterwards, the Synthetic Minority Over-sampling Technique (SMOTE) was used for the data balancing. The two models were trained and evaluated again, in this case with data balancing using the SMOTE method. The results showed that data balancing has positive impact in the performance parameters of the models except the training time and in some cases the accuracy. The accuracy of random Forest model improved with data balancing while that of the LinearSVC dropped with data balancing. Also, the training time can be maintained or minimized by appropriate choice of data balancing approach; whether to balance by adding more data items, dropping some data items or maintaining the total number of data items while adjusting the proportions of each data class in the balanced dataset. Finally, the results showed that the LinearSVC is more suitable for imbalanced dataset while the random Forest is the best model in the case of balanced dataset.

References

1. Zarichuk, O. (2023). Hybrid approaches to machine learning in software development: Applying artificial intelligence to automate and improve processes.
2. Taye, M. M. (2023). Understanding of machine learning with deep learning: architectures, workflow, applications and future directions. *Computers*, *12*(5), 91.
3. Yang, Y., Xia, X., Lo, D., Bi, T., Grundy, J., & Yang, X. (2022). Predictive models in software engineering: Challenges and opportunities. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *31*(3), 1-72.
4. Li, Z., Niu, J., & Jing, X. Y. (2024). Software defect prediction: future directions and challenges. *Automated Software Engineering*, *31*(1), 19.
5. Mishra, R. K., & Raj, G. (2025). Self-Healing AI: An Autonomous Deep Learning Approach for Software Error Correction. *International Journal of Advanced Research in Computer Science and Engineering (IJARCSE)*, *1*(1), 28-34.
6. Lian, B., Kartal, Y., Lewis, F. L., Mikulski, D. G., Hudas, G. R., Wan, Y., & Davoudi, A. (2022). Anomaly detection and correction of optimizing autonomous systems with inverse reinforcement learning. *IEEE Transactions on Cybernetics*, *53*(7), 4555-4566.
7. Pandey, S., & Kumar, K. (2023). Software fault prediction for imbalanced data: a survey on recent developments. *Procedia Computer Science*, *218*, 1815-1824.
8. Goyal, S. (2022). Handling class-imbalance with KNN (neighbourhood) under-sampling for software defect prediction. *Artificial Intelligence Review*, *55*(3), 2023-2064.
9. Goyal, S. (2022). Effective software defect prediction using support vector machines (SVMs). *International Journal of System Assurance Engineering and Management*, *13*(2), 681-696.
10. Ali, M., Mazhar, T., Arif, Y., Al-Otaibi, S., Ghadi, Y. Y., Shahzad, T., ... & Hamam, H. (2024). Software defect prediction using an intelligent ensemble-based model. *IEEe Access*, *12*, 20376-20395.
11. Hairani, H., Widiyaningtyas, T., & Prasetya, D. D. (2024). Addressing class imbalance of health data: a systematic literature review on modified

synthetic minority oversampling technique (SMOTE) strategies. *JOIV: International Journal on Informatics Visualization*, *8*(3), 1310-1318.

12. Elreedy, D., Atiya, A. F., & Kamalov, F. (2024). A theoretical distribution analysis of synthetic minority oversampling technique (SMOTE) for imbalanced learning. *Machine Learning*, *113*(7), 4903-4923.