Project Report

System Defect Prediction Using Machine Learning Techniques

Oluyemi Amujo & Robi Paul

***Abstract*—This report presents our study on the application of machine learning algorithms for predicting system defects. We employed a Microsoft dataset containing information on Windows vulnerabilities and applied methods such as expectation maximization (EM) and recursive feature elimination (RFE) to enhance data quality and model accuracy. Our evaluation encompassed four different ML models: Logistic Regression, LightGBM, Random Forest, and Neural Networks. We found that non-linear models outperformed Logistic Regression, specifically LightGBM, Random Forest, and Neural Networks. LightGBM and Random Forest achieved an AUC value of 70% following hyperparameter optimization. Our research underscores the significance of data preprocessing, including using EM and feature selection through RFE for successful defect prediction. Looking forward, we aim to delve into data normalization and the development of more sophisticated neural network architectures to refine our models further. Our findings offer valuable insights for tackling the complex issue of system defect forecasting using machine learning.**

I. Introduction

System defects are flaws, weaknesses, or vulnerabilities in a system or its components that can cause unexpected or unwanted behavior [1]. Hardware or software problems, malicious software, or other factors can cause these defects [2]. Machine learning techniques have shown promise in predicting system flaws, but they remain an open problem in multiple dimensions [3]. Some of these include determining the best model for a particular scenario, preparing the dataset for the best training performance, and many more.

As a prerequisite for the machine intelligence course, this article discusses work on machine learning algorithms for system defect prediction. Datasets with instances of different system defects, such malware and hardware faults, inspired this. The project aims to evaluate how well various machine-learning algorithms forecast system faults. The two primary contributions—which are rarely discussed in most machine learning articles—are the use of expectation maximization (EM) to fill in missing values in the dataset and the application of recursive feature elimination (RFE) for optimal performance. The GitHub URL provided in this report's appendix provides access to the code and dataset utilized for the project.

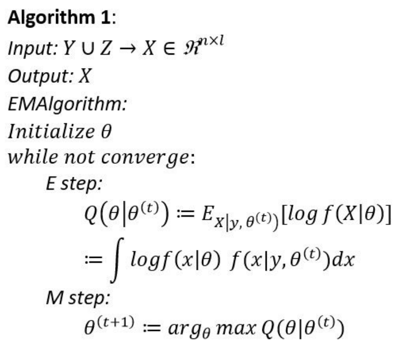
II. Methodology

In this study, we adopt an approach akin to that used by Assim et al. [4], with a notable modification implementing a supervised learning classification technique. Our work utilizes a dataset released by Microsoft Corp. about Windows-based system vulnerabilities and flaws, which is featured in a Kaggle (online platform) competition. We methodically segmented the project into three distinct stages. The initial phase addresses missing data via the Expectation Maximization (EM) algorithm. This is followed by a stage dedicated to selecting fitting features, culminating in a thorough performance analysis. Finally, the performance of four different ML models was investigated to showcase the performance parameters with the improved dataset.

A. Prepossessing and Expectation Maximization (EM)

Initially, the dataset had 81 features, of which about 77 had significant missing values. This necessitated the adoption of EM for missing value imputation.
Considering the general model for EM (1), X is the observed data points, Z is the unobserved or missing values, and θ is a vector of unknown parameters [5]. The idea is that if the value of the parameter θ is known, then the value Z can be found by maximizing the log-likelihood over all possible values of Z either simply by iterating over it or through an algorithm.

$$\theta^{(t+1)} = \arg\max_{\theta} E_{Z\sim p_{\theta}(.|X,\theta^{(t)})}[\log p(X,Z|\theta)] \qquad (1)$$


```
Algorithm 1:
Input: $Y \cup Z \rightarrow X \in \Re^{n\times l}$
Output: $X$
EMAlgorithm:
Initialize $\theta$
while not converge:
    E step:
        $Q(\theta|\theta^{(t)}) \coloneqq E_{X|y,\theta^{(t)}}[\log f(X|\theta)]$
        $\coloneqq \int \log f(x|\theta)\ f(x|y,\theta^{(t)})dx$
    M step:
        $\theta^{(t+1)} \coloneqq arg_{\theta}\ max\ Q(\theta|\theta^{(t)})$
```

where Y, y observations; Y = random variable; y = realization of Y, X, x while Z, z, missing data such that $X(Y,Z).\theta$ is the unknown deterministic parameter. The complexity of this algorithm depends on the number of features that contain missing values.
In our case, iterating over the dataset is inefficient, considering the large size of the missing data. Therefore, given that the dataset D^{hxl} in dimension hxl we split $Dintochunksof$d blocks as defined in (2)

$$d = \frac{D}{n'} \qquad (2)$$

where d^{hxl_i} is a chunk of block in dimension $hxl_i, i = 1, ..., n$. We then efficiently apply EM to each block independently.

B. Recursive Feature Elimination (RFE)

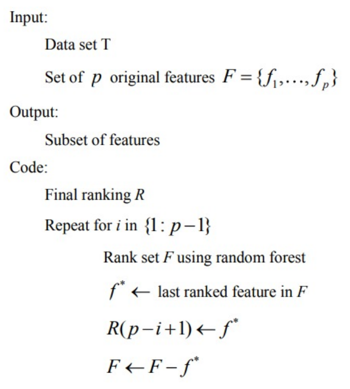
Recursive Feature Elimination (RFE) enhances model efficiency by iteratively discarding less significant features, thus simplifying the model while maintaining key variables. This technique involves building a model with all available features and systematically evaluating and removing the least important ones while maintaining accuracy. RFS effectively reduces the risk of overfitting when this technique is used alongside the Expectation Maximization (EM) algorithm, which improves dataset quality by filling in missing values. Building on this approach, our methodology incorporates the use of the random forest algorithm. Random forest offers an internal mechanism for assessing feature importance, complementing the RFS process. This is achieved by computing feature importance scores within each decision tree that forms part of the random forest ensemble. In these trees, each node is split into two children nodes, with the splitting criterion aimed at reducing the node's impurity. This reduction in impurity is quantified using the Gini importance measure, as described in [6]. This method aligns with the principles of RFS and enhances its effectiveness by providing a robust, internal metric for feature evaluation.

$$i = 1 - \sum_j p(j)^2 \tag{3}$$

$$\Delta i = i_{\text{parent}} - (p_{\text{left}} \cdot i_{\text{left}} + p_{\text{right}} \cdot i_{\text{right}}) \tag{4}$$

$$\alpha \Delta I = \sum_k \Delta i_k \tag{5}$$

These formulae (3-5) indicate the importance of every feature. The greater value indicates that the feature is more important.



```
Input:
    Data set T
    Set of $p$ original features $F = \{f_1, \ldots, f_p\}$
Output:
    Subset of features
Code:
    Final ranking $R$
    Repeat for $i$ in $\{1 : p-1\}$
            Rank set $F$ using random forest
            $f^* \leftarrow$ last ranked feature in $F$
            $R(p-i+1) \leftarrow f^*$
            $F \leftarrow F - f^*$
```

C. Model Description

We adopt the general model (6) to study the logistic regression with high-dimensional space. The goal is to predict y $[1, 0]$ from the dataset u.

$$y = \sigma(w_0 + \sum_{i=1}^{n} w_i u_i) \tag{6}$$

1) LightGBM: LightGBM is a tree-based gradient-boosting model that builds sequential trees where each one amends the errors of its predecessors [7]. This process determines the optimal leaf weights and the maximum Γ_K for a given tree structure $q(x)$.

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{7}$$

$$\Gamma_T^* = -\frac{1}{2} \sum_{j=1}^{J} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} \tag{8}$$

2) Random Forest: Random Forest is another tree-based algorithm that employs multiple decision trees to improve accuracy and control overfitting [8]. Each tree votes for classification or averages for regression, leveraging the diversity. The algorithm provides feature importance metrics, beneficial for interpretability in academic research.

3) Neural Network: Because this work is a binary classification problem, a simplified version of multi-task deep neural networks for software defect prediction [9] is used.

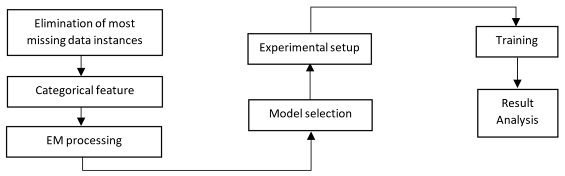


Fig. 1. Working diagram of the Android App

III. Experimental Setup

This study utilized Python 3.9.11 and key machine learning and data analysis libraries, including TensorFlow 2.3, Pandas, and Scikit-learn. TensorFlow enabled the building and training neural network models, while Pandas provided data manipulation capabilities. Scikit-learn implemented traditional machine learning algorithms such as logistic regression random forests and model evaluation metrics. Together, this Python-based framework delivered an effective environment for executing the study's computational workflows in a streamlined manner.

The main idea is to conduct experiments while assigning best-performing values for commonly used parameters to each model (LogisticRegression, LightGBM, Random Forest, NN). To calculate the best values for these parameters, we utilized the Gridseach technique. However, some initial NN parameters are worth mentioning. The following settings: Conv1D (filters = 32, kernel size = 3, activation ='relu' and activation = 'sigmoid' as well as loss = 'binary crossentropy'.

After deleting the features containing 90% missing data or 90% one unique value, we had 55 features. Among these, 32 displayed varying degrees of missing data, as illustrated in Fig 2. Further, an Expectation Maximization (EM) analysis was conducted to impute these missing values.

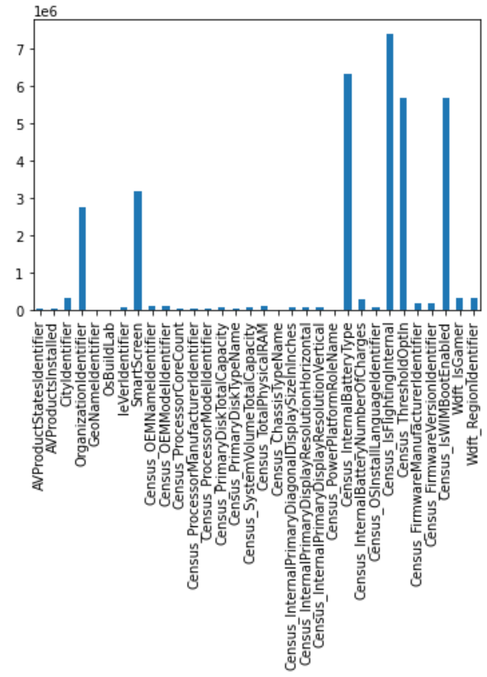


Fig. 2. Missing data distribution

Fig. 3 demonstrates the impact of feature relevance on model accuracy. Using RFE with EM analysis for missing data, the graph (right to left) reveals a progressive decline in accuracy when features are removed from most to least important. Notably, eliminating the 25th reduces accuracy significantly in the context of this model, indicating the significance up to that feature. Therefore, further analysis focuses on these top 25 features.

Fig. 4 shows a bar graph comparing how well different models perform. It is clear from this chart that logistic regression does not do as well as the other models. The main reason is that logistic regression expects a straightforward, direct link between what you put in and the results you get. But our data isn't simple – it has complex patterns that logistic regression can not handle. On the other hand, models like Random Forest and neural networks are built to

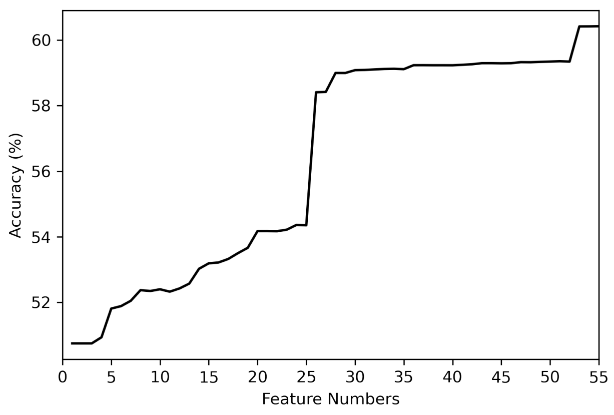


Fig. 3. Results of Recursive Feature Selection (RFE)

deal with this complex data, so they perform better.

Next, both LightGBM and Random Forest have similar results. This is because they both use a method called decision trees, which is good at figuring out complex patterns and how different bits of data affect each other. That is why these models work so well with our data. The neural network model does a bit worse than the tree-based ones but still better than logistic regression. This might be because neural networks need the data to be prepared in a certain way, like making sure everything is scaled correctly. If this is done right, the neural network will work better. Decision tree methods like Random Forest and LightGBM don't need the data to be prepared in a specific way, which can give them an edge.

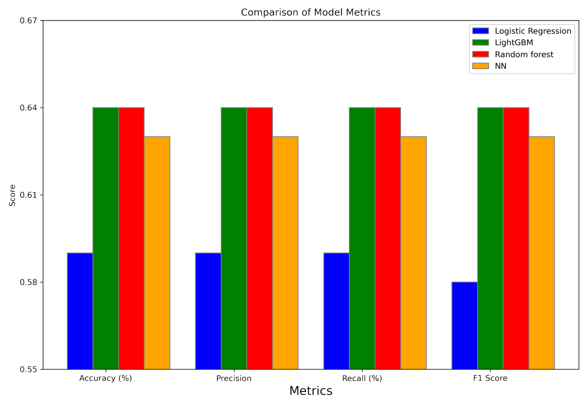


Fig. 4. Bar diagram showing performance parameters

Fig. 5 represents the Roc Curves of our evaluated models, showing a similar trajectory of performance increment to the bar diagram. However, even by tuning the value of generally used hyperparameters, we reached an AUC value of 70 percent with the best-performing model of LightGBM and random forest. Grid search provides a way to optimize hyperparameters but does not guarantee finding the best combination, especially in a high-dimensional hyperparameter space like ours. The best parameters on the training set may not generalize perfectly to unseen data, leading to a plateau in performance metrics like AUC.

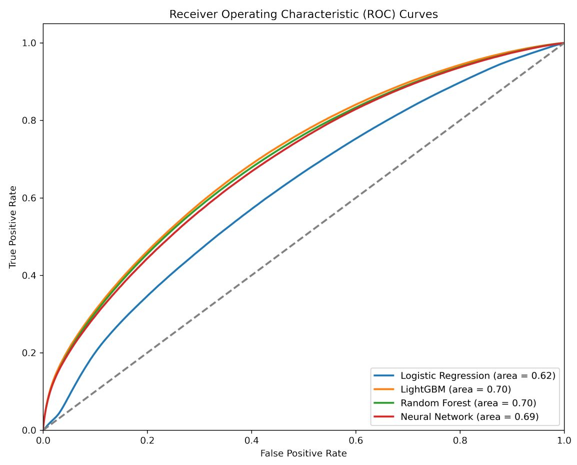


Fig. 5. ROC curve for different model

IV. Conclusions and Future Work

In this work, we have demonstrated three significant aspects of machine intelligence development: 1) the importance of minimizing noise from datasets by using EM, 2) the significance of removing unnecessary features using RFE, 3) the limitation of the linear model (logistic regression) compare to non-linear counterpart (NN, RF, and LighGBM) as seen in performance analysis.

Lastly, we observed a significant variation in the values range among the features, potentially impeding model performance. Future efforts will encompass dataset normalization. Additionally, research has demonstrated that, with adequate training time and batch numbers, neural networks exhibit improved performance as the number of layers increases. Considering this, our future work will entail training the neural network with multiple layers and a substantial number of batches.

Author's Contribution:

Oluyemi Amujo: Conceptual algorithm design and coding for the EM Analysis, report writing and editing.
Robi Paul: Data preprocessing, coding (RFE analysis, Model training, and Tuning), Final data handling (Plotting & Github), Report writing, and Finalizing.

Model & dataset Links

https://github.com/RobiPaul7248/Machine-Intelligence-CMPE67701.2231-.git

References

[1] C. Schmittner, T. Gruber, P. Puschner, and E. Schoitsch, "Security application of failure mode and effect analysis (fmea)," *Computer Safety, Reliability, and Security: 33rd International Conference, SAFECOMP 2014, Florence, Italy, September 10-12, 2014. Proceedings 33*, pp. 310–325, 2014.

[2] V. M. Igure and R. D. Williams, "Taxonomies of attacks and vulnerabilities in computer systems," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 6–19, 2008.

[3] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *Iet Software*, vol. 12, no. 3, pp. 161–175, 2018.

[4] M. Assim, Q. Obeidat, and M. Hammad, "Software defects prediction using machine learning algorithms," *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI)*, pp. 1–6, 2020.

[5] Wikipedia, "Expectation–maximization algorithm wikipedia the free encyclopedia," 2023.

[6] C. Zhang, Y. Li, Z. Yu, and F. Tian, "Feature selection of power system transient stability assessment based on random forest and recursive feature elimination," *2016 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, pp. 1264–1268, 2016.

[7] M. Onoja, A. Jegede, J. Mazadu, G. Aimufua, A. Oyedele, and K. Olibodum, "Exploring the effectiveness and efficiency of lightgbm algorithm for windows malware detection," *2022 5th Information Technology for Education and Development (ITED)*, pp. 1–6, 2022.

[8] B. M. Khammas, "Ransomware detection using random forest technique," *ICT Express*, vol. 6, no. 4, pp. 325–331, 2020.

[9] Q. Huang, Z. Li, and Q. Gu, "Multi-task deep neural networks for just-in-time software defect prediction on mobile apps," *Concurrency and Computation: Practice and Experience*, p. e7664, 2023.