Al-Rafidain Journal of Computer Sciences and Mathematics (RJCM)

www.csmj.mosuljournals.com

Software Defect Prediction Using Extreme Gradient Boosting (XGBoost) with Optimization Hyperparameter

Tariq Najim AL-Hadidi[1]*, Safwan Omar Hasoon[2]

[1,2]*Software Department, College of Computer Sciences and Mathematics, University of Mosul, Iraq*

Emails: tariq2022hadidi@gmail.com, dr.safwan1971@uomosul.edu.iq

Article information

Article history:
Received: 19/8/2023
Accepted: 12/11/2023
Available online: 25/6/2024

Abstract

Software applications have become an integral part of our daily lives, permeating critical domains like traffic control, aviation, and self-driving cars. Software defects can lead to human or material risks. Therefore, ensuring the reliability and quality of software in such systems presents a significant challenge to software companies. To address this challenge, many companies have turned to machine learning, leveraging historical project data, to predict and classify software defects. This study focuses on the classification of software defect prediction using machine learning techniques, particularly classification methods. Among the techniques employed is eXtreme Gradient Boosting (XGBoost), a powerful algorithm for regression and classification analysis based on gradient boosting decision trees (GBoost). XGBoost offers several hyperparameters that can be fine-tuned to enhance model performance. The study employs a hyperparameter tuning approach known as grid search, validated through 10-fold cross-validation. The hyperparameters configured for XGBoost encompass n_estimators, max_depth, subsample, gamma, colsample_bylevel, min_child_weight, and learning_rate. The results of this investigation illustrate that utilizing algorithms with hyperparameter tuning can significantly enhance the XGBoost algorithm's performance, leading to precise and accurate classification of software defects. This advancement holds great promise for improving the quality and reliability of software systems across various critical domains.

Keywords:
Accuracy, eXtreme Gradient Boosting (XGBoost), hyperparameter tuning, precision, software defects

Correspondence:
Author: Tariq Najim AL-Hadidi
Email: tariq2022hadidi@gmail.com

1. Introduction

Machine learning techniques have revolutionized various domains, and software engineering is no exception. Software Defect Prediction using Machine Learning (ML) is a burgeoning field that leverages the capabilities of ML algorithms to proactively identify potential defects in software code, helping developers catch and rectify issues early in the development process. By doing so, it not only improves software quality but also reduces development time and costs.[1].

Usually, various software metrics, including McCabe's metrics, Halstead's metrics, and static code metrics, are commonly employed to build predictive models. These metrics are used to analyze software, assess its quality and characteristics, and aid in enhancing software development, maintenance, and management processes by establishing standardized criteria for software evaluation. Consequently, they are widely used in the majority of software defect prediction models.[2].

In recent times, numerous machines learning classifiers, including decision trees and ensemble learning, have been investigated to enhance software defect prediction prior to the software testing phase. This involves identifying code sections likely to be problematic, where errors are likely to occur.

Predicting defect-prone code segments can guide more targeted testing efforts, reducing overall testing costs and efforts, while enhancing software quality and reliability[2] .
When predicting software defects, it has been found that ensemble machine learning techniques (such as boosting, bagging, and stacking) exhibit higher accuracy and reliability compared to individual classifiers. Optimizing hyperparameters significantly impacts classifier performance. Limited studies using optimization techniques have reported improved prediction performance. Our study aims to address this gap by conducting an investigation utilizing the XGBoost technique: a powerful machine learning algorithm widely recognized and employed across various fields, including software engineering for defect prediction. XGBoost's popularity stems from its ability to handle intricate relationships within data and deliver high predictive performance. Leveraging other machine learning techniques for preprocessing the utilized dataset and employing hyperparameter optimization for fine-tuning further enhances its effectiveness. Our study seeks to fill this research gap by utilizing XGBoost and optimizing its hyperparameters for improved defect prediction performance in software engineering [3].

The rest of the paper is organized as follows. Section 2 provides background information on the significance of the XGBoost algorithm used for software defect prediction. Section 3 examines prior work in ensemble learning and hyperparameter tuning for software defect prediction. The experimental study process is detailed in Section 4. Results and discussion are presented in Section 5. Finally, Section 6 summarizes the conclusions and outlines potential future work.

2. Related Work

Researcher Iqbal and others developed a framework for predicting software defects using clustering classification and feature selection techniques. The framework includes three main stages to form a set of different classifiers using clustering classification techniques, selecting the most important features using the integrated feature selection technique, and merging the results of the different classifiers; Two different dimensions are used in the framework, one: with feature selection, and the second: without feature selection, and each dimension used two grouping techniques with the Random Forest classifier: Bagging and Boosting, and the framework is evaluated using a set of data sets. The results showed an improvement in the accuracy of predicting software defects and a reduction in the number of features used[4].

Researchers Khuat and Le compared the effectiveness of combining different sampling methods and ensemble learning through two learning schemes on an imbalanced software defects data set, and the results obtained indicated that the balanced training data set contributes to a significant improvement in the performance of each of the ensemble models. And the basic classifiers compared to those that use the original unbalanced dataset. In the first method: each basic classifier is trained on a different balanced dataset, while in the second method, all basic learners are trained on a single balanced dataset. The methodology consists of three components: Data balancing, classifier training, and classification[5].

Researcher Suresh Kumar and others presented a new model for predicting software defects using an assembly learning technique, or aggregating Bootstrap or Bagging. The performance of the model was compared with established machine learning algorithms such as: decision trees, K-nearest Neighbor, Support Vector Machine, And others using a specific data set. The proposed bagging approach has shown superior performance over other methods in predicting software defects. The bagging method achieved an accuracy rate exceeding 95% on multiple data sets, superior to other models[6].

Researcher Yang and others used various machine learning techniques, specifically using the stacking learning approach; This new approach was developed to build better models and improve evaluation methods, and in addition, the need for appropriate pre-processing work on software defect data[7].

Researcher Ibrahim and others used the ELFF dataset, which is based on 23 open source Java projects in which the researchers used different ensemble learning algorithms, such as: AdaBoost, Gradient Boost, Bagging, Random Forest, and their balanced versions, to build a software defect prediction model on the ELFF dataset[8].

3. Background of the Research

A final predictive model is constructed through the integration of a diverse set of machine learning classifiers using a technique known as ensemble learning in machine learning. These ensembles possess the capability to explore complex data patterns necessary for achieving accurate classification and making relevant decisions. The process of ensemble learning consists of two fundamental stages:

1. **Base Classifier Training:** This stage involves training a set of individual machine learning classifiers, which are collectively assembled to form the base classifier ensemble.
2. **Creation of the Final Model:** This is achieved by combining the outputs of the base classifiers using methods such as averaging or voting [9].

When base classifiers employ a single classification algorithm, the ensemble is classified as homogeneous. Conversely, when a diverse set of methods is used, it is referred to as heterogeneous [10].
The focus of this study revolves around the application of ensemble learning based on trees, which leverages homogeneous ensembles, culminating in a final predictive model achieved through the integration of multiple base classifiers.
In essence, this study focuses on the integration of machine learning classifiers through homogeneous ensemble learning methodologies to effectively predict software defects. This comprehensive approach harnesses various techniques, each

contributing to enhancing predictive accuracy and mitigating common challenges in software defect prediction.

3. Boosting

Boosting is a powerful ensemble learning technique in machine learning. It's used to improve the accuracy and performance of classification algorithms, and it works by combining the predictions of multiple weak or base learners (typically decision trees) to create a strong, accurate predictive model [11][12][13].

Here's a more detailed explanation of boosting:

1. **Ensemble Learning:** Boosting is a type of ensemble learning, where multiple models are combined to produce a single, robust predictive model. The key idea is that by combining the strengths of several weak learners, the ensemble model can achieve better predictive performance than any individual model.
2. **Weak Learners:** Boosting focuses on using weak learners as base models. Weak learners are classifiers that perform slightly better than random guessing, such as simple decision trees with limited depth. These are often called "stumps" because they are very shallow trees.
3. **Iterative Approach:** Boosting is an iterative process. Initially, each data point is given equal weight, and a weak learner is trained to minimize the error. After each iteration, the weights of misclassified data points are increased so that the next learner focuses more on the previously misclassified examples. This process continues until a predefined number of iterations is reached or until the model's performance plateaus.
4. **Weighted Voting:** During prediction, each weak learner contributes to the final prediction, but their influence depends on their individual performance. Weak learners that perform better have more weight in the final prediction, while those that perform worse have less influence.
5. **Usefulness:** Boosting is highly useful in scenarios where high predictive accuracy is required. It's widely used in various machine learning applications, including:
    - **Classification:** Boosting is commonly used for classification tasks, where it can significantly improve the accuracy of models, making it valuable in applications like spam email detection, image classification, and medical diagnosis.
    - **Regression:** Boosting techniques can also be adapted for regression problems, where the goal is to predict a continuous numeric value rather than a category.
6. **Popular Boosting Algorithms:** There are several popular boosting algorithms, and one of them is XGBoost (eXtreme Gradient Boosting), which you mentioned earlier. XGBoost is known for its efficiency, scalability, and high predictive accuracy. Other notable boosting algorithms include AdaBoost, Gradient Boosting, and LightGBM.
7. **Hyperparameter Tuning:** Like many machine learning algorithms, boosting algorithms often have hyperparameters that can be fine-tuned to optimize performance. Hyperparameter tuning involves adjusting parameters such as the learning rate, maximum depth of trees, and the number of iterations to achieve the best results.

4. Extreme Gradient Boosting (Xgboost)

The XGBoost technique is an advancement of the gradient boosting method introduced by Dr. Tianqi Chen from the University of Washington in 2014. Gradient boosting is an algorithmic approach capable of finding optimal solutions for a variety of problems, notably in regression, classification, and ranking. The core concept of this algorithm involves iteratively adjusting learning parameters to minimize the loss function (a mechanism for evaluating model performance) [2]. Enhanced Gradient Boosting Decision Trees (GBDTs) harness the intelligence of aggregated predictions from individual decision trees to yield improved comprehensive predictions. The boosting process entails training multiple weak decision trees in successive steps to enhance prediction. A weak tree model might exhibit good performance only on a portion of the training dataset. By judiciously combining multiple weak learners, an exceptionally robust ensemble model is crafted. XGBoost is a popular technique for GBDT, demonstrating superior performance across a range of data science problems with precision and speed. Its training continues iteratively by introducing new decision trees that predict the errors of previous trees, which are then combined with the previous trees to produce the final prediction, as illustrated in the following **Fig. 1**:

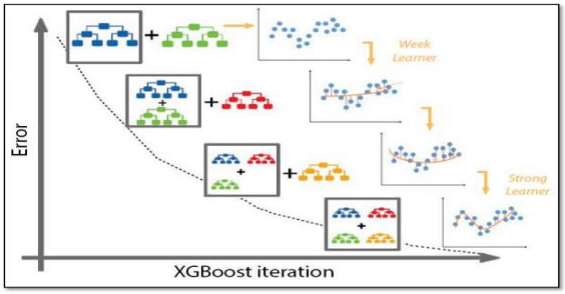

Fig. 1. Error Reduction through Progressive Training Using Extreme Gradient Boosting Algorithm[14].

The XGBoost Algorithm Steps [15]:

Step 1: Calculate Residual (New Objective) Calculate the residual Ri for all samples in the target variable y:

$$\text{av} = \text{average (yi)} \quad \dots (1)$$

$$\text{Ri} = \text{yi} - \text{av} \quad \dots (2)$$

Where av represents the average of sample values and Ri represents the residual of samples from the target variable y.

Step 2: Create a New Decision Tree Build an optimal decision

tree from the feature set f and the residual Ri.

Step 3: Calculate New Residual Based on the new errors DTi, update the new residual $Ri + 1$, which is derived from learning from the errors (residual Ri) of the previous tree, and the learning rate α:

$$\text{Ri} + 1 = \text{av} + \alpha \times \text{Ri} \quad \ldots(3)$$

Where α= [0.01, 0.1, 0.001,0.3] represents the learning rate.

Step 4: Boosting Repeat steps 2 and 3 until all trees are trained. For new data samples, XGBoost makes predictions by sequentially considering steps on all decision trees DTi.

Fig. 2 illustrates the schematic representation of the XGBoost algorithm..

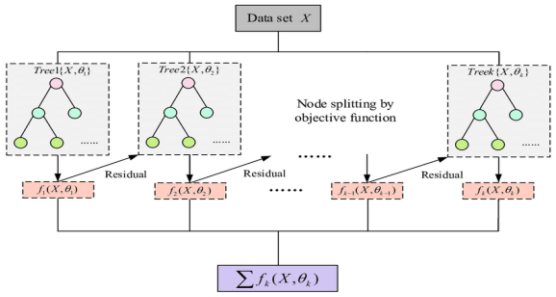

Fig. 2. Depicts the schematic representation of the XGBoost algorithm.

5. Hyperparameter Tuning Dengan Grid Search Cross Validation

In machine learning techniques, there are several parameter values expected to enhance model performance, known as hyperparameters. Hyperparameters are used to optimize algorithm performance, significantly influencing various testing models. Hyperparameters are effectively executed by either manual exploration or predefined limited testing of hyperparameter sets. The search for hyperparameters can be done manually or through testing a predefined set of hyperparameters on specific settings. One of the hyperparameter techniques that will be applied is Cross-Validation (CV), which will optimize and estimate the following hyperparameters to enhance model performance in classification [16]. Specifically, there are 7 hyperparameters, as depicted in **Table 1** below:

Table 1. Optimal Hyperparameter Values for Best Results

Hyperparameter	Uses of Hyperparameters
n_estimators	The number of trees used for the classification process
max_depth	The inner level of the tree
min_child_weight	Bobot minimal
eta (learning_rate)	Helps streamline steps in model updates
gamma	Minimize loss reduction
subsample	Instance ratio of the training data
colsample_bylevel	Ratio of the training data used to create the tree

6. Evaluation

In general, the performance of a classification algorithm is evaluated by comparing the expected value of the classification algorithm with the target value of the test data variable as actual data. There are several methods to evaluate the obtained classification model, including accuracy, precision, and recall. The performance evaluation value of the XGBoost model is obtained from the confusion matrix. The confusion matrix is a measurement tool in the form of a matrix, from which various evaluation values such as accuracy, precision, and recall are derived.

7. Experimental Study

In this section, we discuss a novel approach to address the issue of imbalance in the software defect prediction dataset. The approach involves oversampling the minority class using oversampling techniques and utilizing data preprocessing to prepare the data for training an XGBoost model for defect prediction. This is achieved through optimizing parameter values using an optimizer, which in turn enhances the model's performance, ultimately improving testing efficiency. **Fig. 3** illustrates the mechanism employed in this study.

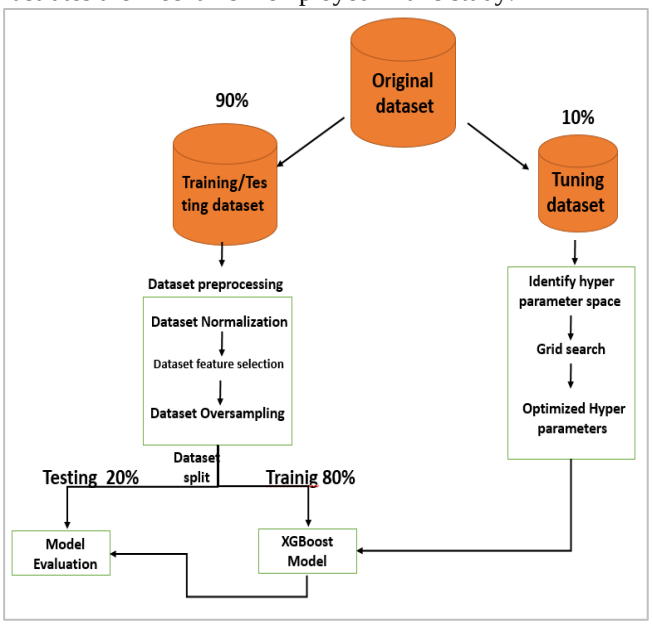

Fig. 3. Research Methodology.

1. Data Preprocessing: Predictive defective datasets are valuable resources in software engineering research, containing data from previous software projects, including software metrics and defect information. Researchers and practitioners utilize these predictive defect datasets to develop predictive models that forecast the likelihood of defects occurring in software units or projects. This is achieved through the use of machine learning and statistical techniques to analyze the relationships between software metrics and defects. These models help identify defect-prone areas early in the development process, improve software quality, and reduce error-fixing costs [17]. In this study, (5) datasets from[18]. the NASA MDP repository will be utilized for open-source predictive defect analysis. These datasets are publicly available and include crucial features (software metrics), as shown in **Table 2**.

Table 2. Overview of NASA MDP repository datasets.

Dataset	Programming Language	Instances	Non-Defective Units	Defective Units
CM1	C	498	449	49
MC1	C & C++	9466	9398	68
PC1	C	1109	1032	77
PC4	C	1458	1380	178
KC1	C++	2109	1783	326

Data cleaning is a crucial step in machine learning, as datasets often contain erroneous, duplicate, or misclassified data. Inaccurate data can lead to unreliable results and algorithms, rendering them untrustworthy. Therefore, data cleaning plays a vital role in algorithm development. It can be defined as the process of correcting or removing missing or unnecessary values from a dataset before analysis[19].

2. Relevant feature engineering is undertaken to capture program characteristics and patterns that contribute to defect prediction. Consider techniques like dimensionality reduction (e.g., Principal Component Analysis) to reduce noise and enhance feature representation. Data will be partitioned into two segments during this stage: training data and testing data, facilitating data exchange.
3. Dataset Balancing: In this stage of preprocessing, dataset balancing is performed in machine learning, aiming for an even distribution of data across classes (defective and non-defective) within the dataset. Data imbalance occurs when we have a differing number of samples for each class, a common issue in the field of machine learning due to irregularly balanced classes. Imbalance negatively impacts the performance of machine learning models, causing them to focus more on overrepresented classes while neglecting the underrepresented ones. For instance, in defect prediction, it is common for most data instances to belong to the non-defective class, potentially hindering machine learners' performance, as they tend to maximize predictive accuracy by disregarding the minority class [12].

   The problem of data imbalance can be mitigated through resampling techniques, where samples from the training data are either added or removed to achieve a more balanced distribution. Often, a combination of various methods is employed. Data resampling can be performed in several ways, including:

   A. Oversampling: This involves increasing the number of samples by adding instances of the minority class. New instances of the minority class are often generated by replicating existing minority instances.

   B. Undersampling: This is done by removing instances belonging to the majority class, resulting in a reduction of the dataset size. Undersampling typically involves randomly discarding instances from the majority class [12].
4. Perform hyperparameter tuning using Cross-Validation (CV) grid search.
5. Applying XGBoost: In this stage, the XGBoost algorithm is applied to the data under study. Simulations of classification using XGBoost are conducted with reduced quantities of data. The stages to be executed are as follows:
6. Train the XGBoost model using the training data.
7. Conduct prediction using the pre-trained XGBoost model.
8. The model must be evaluated using two metrics, as they have been a fundamental requirement to determine its proper functioning. AUC, accuracy, precision, ROC curve, and F1-score are all evaluation metrics. To start, the definition of the confusion matrix is given, as shown in **Fig. 4** and **Table 1**: Interpretation of Results and Conclusions.

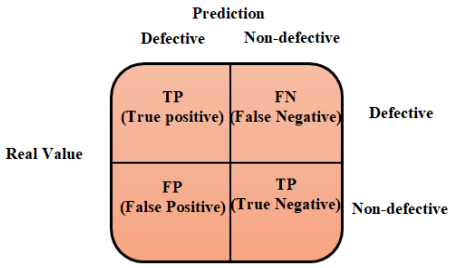

Fig. 4. Confusion Matrix.

8. The Confusion Matrix and Evaluation Metrics

The Confusion Matrix is a table that displays the results of classification predictions. It summarizes the correct and incorrect predictions by comparing them with the actual values. It consists of four categories based on the comparison of predicted values and actual values, which are described as True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) ([20]).

1. True Positive (TP): It represents the correctly predicted positive instances of a software defect. In other words, when a defect is present, and the prediction correctly identifies it as a defect.
2. True Negative (TN): It represents the correctly predicted negative instances of a software defect. In other words, when no defect is present, and the prediction correctly identifies it as not a defect.
3. False Positive (FP): It represents the incorrectly predicted positive instances of a software defect. In other words, when no defect is present, but the prediction incorrectly identifies it as a defect.
4. False Negative (FN): It represents the incorrectly predicted negative instances of a software defect. In other words, when a defect is present, but the prediction incorrectly identifies it as not a defect.[21].

and evaluation metrics

1. **Accuracy**: Accuracy is the total number of correct predictions divided by the total number of predictions made on the dataset. The best accuracy is 1, while the worst is 0. It can be calculated using the formula (1):

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \quad (1$$

2. **Precision**: Precision is the ratio of true positive predictions (TP) to the total number of positive predictions. The best precision is 1, and the worst is 0. It can be calculated using formula (2):

$$\text{precession} = \frac{TP}{TP + FP} \quad (2)$$

3. **Recall**: Recall, also known as sensitivity or true positive rate, is the ratio of true positive predictions (TP) to the total number of actual positives (TP + FN). It can be calculated using formula (3):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

4. **F1-Score**: The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It can be calculated using formula (4):

$$\text{F1} - \text{score} = 2 * \frac{\text{precession } * \text{Recall}}{\text{precession } + \text{Recall}}$$

9. Results And Discussion

The performance results of the XGBoost algorithm using default parameter values are presented using performance metrics such as accuracy, precision, recall, and F1-score in the tables below.

Table 3 illustrates the classification results of the defect prediction dataset without hyperparameter tuning. The XGBoost method is applied to classify the defect prediction dataset. The first step is to remove unwanted variables in the study. Subsequently, missing and outlier values in the data are identified. During the modeling phase, the dataset is divided into training and testing data. The training data constitutes 80% of the total dataset, while the testing data accounts for the remaining 20%.

In this study, two experiments are conducted. The first experiment involves classification using XGBoost without hyperparameter tuning, and the second experiment involves classification using XGBoost with hyperparameter tuning through the grid search method. The classification is performed using the XGBoost algorithm with the assistance of the XGBoost library in Python.

In the classification experiments without hyperparameter tuning, 80% of the training data is used to train the XGBoost model. The testing data is then used to measure the performance of the resulting model in terms of accuracy, precision, and recall. The performance results of the XGBoost model without hyperparameter tuning can be observed.

The outcomes of these experiments provide insights into the effectiveness of the XGBoost algorithm in defect prediction tasks, both with and without hyperparameter tuning. The evaluation metrics will help assess the model's performance and guide the discussion of its implications and conclusions.

Table 3. Performance Results of XGBoost Algorithm Using Default Parameters

Datasets	Accuracy	Precision	Recall	F1-score
PC1	0.933	0.927	0.942	0.934
PC4	0.938	0.920	0.961	0.94
KC1	0.7433	0.756	0.728	0.742
MC1	0.975	0.952	0.998	0.975
CM1	0.888	0.844	0.938	0.888

While **Table 4** illustrates the performance results of the XGBoost algorithm for each dataset, achieved by tuning the hyperparameters of the XGBoost algorithm.

Table 4. Presents the performance results of the XGBoost algorithm using the tuning of the hyperparameters to achieve the best values.

Datasets	Accuracy	Precision	Recall	F1-score
PC1	0.963	0.921	0.985	0.95
PC4	0.961	0.933	0.995	0.963
KC1	0.795	0.816	0.784	0.8
MC1	0.992	0.984	1	0.991
CM1	0.938	0.922	0.953	0.951

From **Fig. 5**, it can be observed that the accuracy or the percentage that defines the similarity between the predicted results of software defect prediction using the XGBoost algorithm with the actual measured test data has led to the classification being conducted using the XGBoost algorithm with the process of hyperparameter tuning, which involves optimizing the parameters. This is beneficial for enhancing the model's performance in classification. There are 7 parameters that are expected to improve the model's performance in classification using the XGBoost method. The hyperparameters tuning is performed on these 7 parameters using the grid search method. The grid search method is considered an accurate approach because when determining the best hyperparameters, each parameter is explored by specifying its prediction value type first. The optimal configuration for the hyperparameter of the grid search is determined based on the highest value of cross-validated accuracy for the hyperparameter candidate. The results of the best hyperparameter values are as follows:

Table 5. Optimal Hyperparameter Tuning for XGBoost Algorithm.

Hyperparameter Grid	Search Values Best	Hyperparameter Values
n_estimator	400, 300, 200, 100	1000
max_depth	8, 7, 6, 5, 4	6
min_child_weight	0, 1, 2, 3, 4, 5, 6, 7	7
eta (learning_rate)	0,3, 0,2, 0,1, 0,05, 0,025	0.3
gamma	0, 0,1, 0,2, 0,3, 0,4, 1, 1,5 ,2	2
subsample	1, 0,75, 0,5, 0,15	1
colsample_bylevel	0.1, 0.2, 0.25, 1.0	1

It is possible to observe the best parameters and their corresponding values that can enhance the performance of the XGBoost algorithm in **Table 5**. Then, this optimal hyperparameter configuration is utilized to retrain the XGBoost model using the entire training dataset. Subsequently, the XGBoost model with tuned hyperparameters is evaluated using the test data to measure its performance in terms of accuracy, precision, and recall.

Regarding the performance results obtained from the XGBoost model with tuned hyperparameters, they are presented in graphical form in **Fig. 5**. It can be noted that the XGBoost model with tuned hyperparameters demonstrates significant improvement. These studies also provide evidence that the pre-training hyperparameter tuning process can enhance algorithm performance, especially for classification techniques.

From the description above, it can be concluded that hyperparameter tuning is recommended as a crucial step before classification, as both studies indicate its positive impact on algorithm performance.

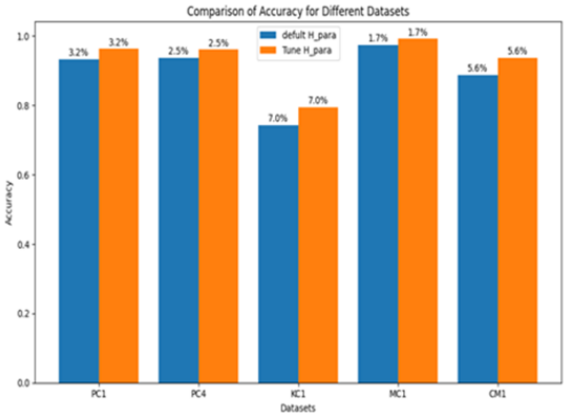

Fig. 5. Accuracy values before and after tune Hyperparameters.

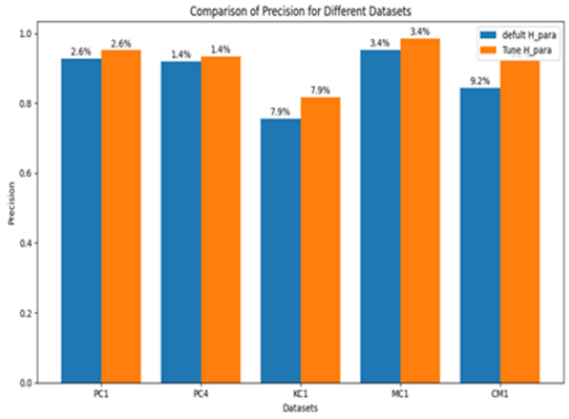

Fig. 6. Precision values before and after tune Hyperparameters.

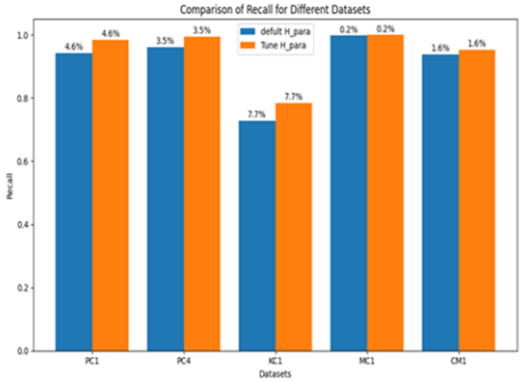

Fig. 7. Recall values before and after tune Hyperparameters

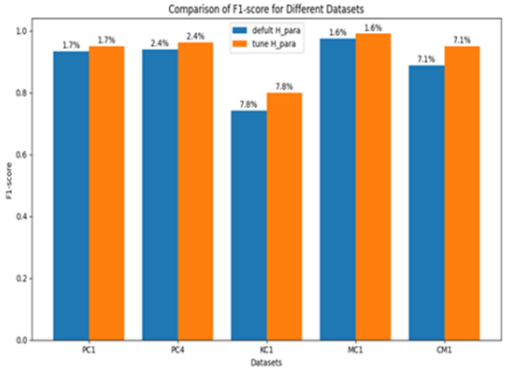

Fig. 8. F1-score values before and after tune Hyperparameters

10. Conclusion

Based on the research findings from the discussion, it can be deduced that the classification results using the XGBoost method with default parameters on the software defect prediction dataset have yielded a model considered to be very good. The model's accuracy, as indicated in **Table 3**, can be classified as falling within the category of good classification. As for the second experiment involving optimization techniques, specifically the process of hyperparameter tuning using 7 hyperparameters through cross-validation, the results of hyperparameter tuning resulted in a model accuracy as presented in **Table 4**, with classification outcomes falling into the "good" classification category. With these outcomes, it has been demonstrated that hyperparameter tuning is the optimal solution if you aim to enhance the performance of the XGBoost algorithm in classification tasks. Adopting other methods to adjust parameters and balance data. Parallel data collection and hyperparameter optimization are important steps for developing accurate prediction models.

Acknowledgement

The authors would express they're thanks to the College of Computer Science and Mathematics, University of Mosul, for supporting this research.

References

[1] T. Menzies, R. Krishna, and D. Pryor, “e Promise Repository of Empirical So ware Engineering Data. hp,” *openscience. us/repo. North Carolina State Univ. Dep. Comput. Sci.*, 2015.

[2] T. Zhang, Q. Du, J. Xu, J. Li, and X. Li, “Software defect prediction and localization with attention-based models and ensemble learning,” *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 2020-Decem, pp. 81–90, 2020, doi: 10.1109/APSEC51365.2020.00016.

[3] H. Yang and M. Li, “Software defect prediction based on smote-tomek and xgBoost,” in *International Conference on Bio-Inspired Computing: Theories and Applications*, 2021, pp. 12–31.

[4] A. Iqbal, S. Aftab, I. Ullah, M. S. Bashir, and M. A. Saeed, “A feature selection based ensemble classification framework for software defect prediction,” *Int. J. Mod. Educ. Comput. Sci.*, vol. 11, no. 9, p. 54, 2019.

[5] T. T. Khuat and M. H. Le, “Evaluation of sampling-based ensembles of classifiers on imbalanced data for software defect prediction problems,” *SN Comput. Sci.*, vol. 1, no. 2, p. 108, 2020.

[6] P. Suresh Kumar, H. S. Behera, J. Nayak, and B. Naik, “Bootstrap aggregation ensemble learning-based reliable approach for software defect prediction by using characterized code feature,” *Innov. Syst. Softw. Eng.*, vol. 17, no. 4, pp. 355–379, 2021.

[7] Z. Yang, C. Jin, Y. Zhang, J. Wang, B. Yuan, and H. Li, “Software Defect Prediction: An Ensemble Learning Approach,” in *Journal of Physics: Conference Series*, 2022, vol. 2171, no. 1, p. 12008.

[8] A. M. Ibrahim, H. Abdelsalam, and I. A. T. F. Taj-Eddin, “Software Defects Prediction At Method Level Using Ensemble Learning Techniques,” *Int. J. Intell. Comput. Inf. Sci.*, vol. 23, no. 2, pp. 28–49, 2023.

[9] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, “A survey on ensemble learning,” *Front. Comput. Sci.*, vol. 14, pp. 241–258, 2020.

[10] S. R. Safavian and D. Landgrebe, “A survey of decision tree classifier methodology,” *IEEE Trans. Syst. Man. Cybern.*, vol. 21, no. 3, pp. 660–674, 1991.

[11] Y. Freund, “Boosting a weak learning algorithm by majority,” *Inf. Comput.*, vol. 121, no. 2, pp. 256–285, 1995.

[12] F. S. De Menezes, G. R. Liska, M. A. Cirillo, and M. J. F. Vivanco, “Data classification with binary response through the Boosting algorithm and logistic regression,” *Expert Syst. Appl.*, vol. 69, pp. 62–73, 2017.

[13] M.-J. Kim, D.-K. Kang, and H. B. Kim, “Geometric mean based boosting algorithm with over-sampling to resolve data imbalance problem for bankruptcy prediction,” *Expert Syst. Appl.*, vol. 42, no. 3, pp. 1074–1082, 2015.

[14] “XGBoost vs LightGBM: How Are They Different.” https://neptune.ai/blog/xgboost-vs-lightgbm (accessed Sep. 06, 2023).

[15] S. M. Kasongo and Y. Sun, “Performance Analysis of Intrusion Detection Systems Using a Feature Selection Method on the UNSW-NB15 Dataset,” *J. Big Data*, vol. 7, no. 1, p. 105, 2020, doi: 10.1186/s40537-020-00379-6.

[16] S. Putatunda and K. Rama, “A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost,” in *Proceedings of the 2018 international conference on signal processing and machine learning*, 2018, pp. 6–10.

[17] Nitin, K. Kumar, and S. S. Rathore, “Analyzing ensemble methods for software fault prediction,” in *Advances in Communication and Computational Technology: Select Proceedings of ICACCT 2019*, 2021, pp. 1253–1267.

[18] “PROMISE Software Engineering Repository.” Accessed: Aug. 21, 2023. [Online]. Available: http://promise.site.uottawa.ca/SERepository/

[19] V. Kumar and C. Khosla, “Data Cleaning-A thorough analysis and survey on unstructured data,” in *2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2018, pp. 305–309.

[20] A. K. Santra and C. J. Christy, “Genetic algorithm and confusion matrix for document clustering,” *Int. J. Comput. Sci. Issues*, vol. 9, no. 1, p. 322, 2012.

[21] Ž. Vujović, “Classification model evaluation metrics,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 6, pp. 599–606, 2021.