Journal of Electronics, Electromedical Engineering, and Medical Informatics
Multidisciplinary: Rapid Review: Open Access Journal **Vol. 6, No. 4, October 2024, pp: 416-428; eISSN: 2656-8632**

RESEARCH ARTICLE **OPEN ACCESS**

Manuscript received May 5, 2024; revised August 2, 2024; accepted August 5, 2024; date of publication October 20, 2024
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v6i4.466

How to cite: Muhammad Noor, Radityo Adi Nugroho, Setyo Wahyu Saputro, Rudy Herteno , and Friska Abadi, "Optimization of Backward Elimination for Software Defect Prediction with Correlation Coefficient Filter Method ", Journal of Electronics, Electromedical Engineering, and Medical Informatics, vol. 6, no. 4, pp. 397-404, October 2024.

Optimization of Backward Elimination for Software Defect Prediction with Correlation Coefficient Filter Method

Muhammad Noor, Radityo Adi Nugroho, Setyo Wahyu Saputro, Rudy Herteno, and Friska Abadi

Computer Science Department, Lambung Mangkurat University, Banjar Baru, South Kalimantan, Indonesia'
Corresponding author: radityo.adi@ulm.ac.id

ABSTRACT Detecting software defects is a crucial step for software development not only to reduce cost and save time, but also to mitigate more costly losses. Backward Elimination is one method for detecting software defects. Notably Backward Elimination may remove features that may later become significant to the outcome affecting the performance of Backward Elimination. The aim of this study is to improve Backward Elimination performance. In this study, several features were selected based on their correlation coefficient, with the selected feature applied to improve Backward Elimination final model performance. The final model was validated using cross validation with Naïve Bayes as the classification method on the NASA MDP dataset to determine the accuracy and Area Under the Curve (AUC) of the final model. Using top 10 correlation feature and Backward Elimination achieve an average result of 86.6% accuracy and 0.797 AUC, while using top 20 correlation feature and Backward Elimination achieved an average result of 84% accuracy and 0.812 AUC. Compare to using Backward Elimination and Naïve Bayes respectively the improvement using top 10 correlation feature as follows: AUC:1.52%, 13.53% and Accuracy: 13%, 12.4% while the improvement using top 20 correlation feature as follows: AUC:3.43%, 15.66% and Accuracy: 10.4%, 9.8%. Results showed that selecting the top 10 and top 20 feature based on its correlation before using Backward Elimination have better result than only using Backward Elimination. This result shows that combining Backward Elimination with correlation coefficient feature selection does improve Backward Elimination's final model and yielding good results for detecting software defects.

INDEX TERMS Backward Elimination, Correlation Coefficient, Software Defect Prediction.

I. INTRODUCTION

In the software development lifecycle finding software defect at during testing stage is highly important. developers should ideally find all software defects, during this stage[1]. Once a flawed system is implemented, it becomes substantially more expensive to identify and rectify defects within the system compared to during the software development phase[2]. Therefore, to find defects more efficiently, software defect prediction is used. Software defect prediction helps reduce cost and save time, also to mitigating more costly losses [2]. Money and time spent on fixing software can be instead be spend improving core functionality. Defects can range from impeding correct function of code, to introduction of vulnerabilities which may be exploited by malicious entities. Therefore, predicting unknown defects in software can assist developers and programmers in obtaining critical information about the type and location of defects to enhance the availability and reliability of the software. Currently, machine learning techniques are considered the most promising for predicting defects in software[3], [4].

Some factors that affect the performance of software defect prediction is redundant data, and irrelevant features in the dataset. Feature Selection (FS) is one way to address this issue. The objective of feature selection is to determine relevant features while removing irrelevant and redundant ones [5], [6]. Feature Selection methods are categorized into three types: Embedded, Filter, and Wrapper. Wrapper methods produces more accurate results than the Filter methods, but the performance of the Wrapper method depends on its specific algorithm implementation. It notably requires more computational power than Filter methods[7].

Wrapper feature selection methods selects the feature subset based on a given criteria. Wrapper methods determines the best subset of features using machine learning, features selected based on the accuracy result of a classification algorithm, wrapper method can be sequential or random [8], [9]. Filter feature selection assigns a scoring value to each feature using statistical measures. Feature weighting is a technique that assigns weights to different features based on their relevance or importance in a machine learning model. By assigning each feature a weight, it can be used for feature selection using the filter method to select subset features [10].

Backward Elimination (BE) is an example of a wrapper feature selection method. Backward Elimination removes one feature at a time that is considered irrelevant to the outcome of the model[11]. The feature selection process repeats until the result meets the parameter set by user, the parameter usually is set if the result become significant. The combination of several features can yield significant results, however the effectiveness of this varies for given combinations. Notably, when Backward Elimination removes a feature, it cannot be added back into the final model. These removed features may potentially improve the performance of the final model if added or present [12]. This drawback is referred to as Omitted Variable Bias [13].

Correlation coefficient is an example of a filter feature selection method. Correlation coefficient models calculate the level of relevance of features based on their correlation, which can be converted into weights to be ranked [14]. Correlation-based feature selection (CFS), or feature selection based on correlation coefficient, is consider by some to be the best-performing feature selection technique in previous research[15], [16].

The study by Balogun et al. [17] introduce the hybrid feature selection (HFS) method that they called rank aggregation-based hybrid multifilter wrapper feature selection (RAHMFWFS). It is used for selection of relevant and irredundant feature from software defect prediction dataset. HFS method is a combination of Wrapper feature selection and Filter feature selection. Balogun et al. combine rank aggregation-based multifilter feature selection (RFMS) method and enhanced wrapper feature selection (EWFS). They found that the RAHFWFS method had a greater positive impact on prediction performance of Naïve Bayes and Decision Tree with the following result Naïve Bayes with average accuracy value of 82.67%, AUC value of 0.802 and Decision Tree average accuracy result value of 83.8%, AUC value of 0.732

The study by Zhang et al.[18] introduced a new feature selection method that they refer to as a large margin hybrid algorithm for feature selection (LMFS), LMFS combines filter method and wrapper method approaches. They used a weighted bootstrapping search to find subset candidates, then a distance-based evaluation to optimize the selection from all the subset candidate and find the final model bases on that. The result of their method is it performed better for classification and model interpretation compare to only using a wrapper method or filter method only.

The study by Shantal et al. [19] proposed a method that they call correlation coefficients with Min-max weighted (CCWP) for data preprocessing. This method was used for improving the performance primarily model accuracy. They used K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Logistic Regression (LR), Neural Network (NN), and Naive Bayesian classifiers (NB) to evaluate their model focusing on its accuracy. They found that only the accuracy results of KNN as classifier were unsuitable, while other classifiers method that were used in the study to evaluate the model outperformance the normal Min-Max normalization method in 40% of the dataset that they used. However other weighting and normalization method outperformed CCWP in most cases.

In recent study by Rahman et al. [20] proposed a method they call multi correlation-based feature selection (MCFS). MCFS integrates two feature selection techniques: Correlation-Based Feature Selection (CFS) and Correlation Matrix-Based Feature Selection (CMFS). This method aims to reduce data dimensionality, eliminate noisy attributes, and enhance the predictive performance of the Particle Swarm Optimization (PSO) algorithm. The study found that PSO combined with MCFS outperformed other methods, achieving an average AUC of 0.891. In contrast, when PSO was applied independently, the AUC was 0.827. Additionally, the combination of PSO with CFS improved the AUC to 0.836, while combining PSO with CMFS resulted in a further increase to 0.839.

Previous studies show that combining feature selection can yield better result and improve its performance. The aim in this study to improve performance of Backward Elimination for detecting software defect, features of the dataset will be selected, with subsequent application of Backward Elimination. The features will be determined using correlation coefficients to assess their relevance as shown from previous studies using correlation coefficients to determined relevant feature can improve performance. Utilization of using Backward Elimination in conjunction Correlation feature selection filter method for software defect prediction has not been attempted, it is anticipated that by using alternative approaches to evaluate the relevance of features and removal of identified irrelevant features, the limitations of Backward Elimination can be mitigated, improving the prediction performance of the final Backward Elimination model. The findings of this study are expected to provide contributions such as:

1. Introducing another filter feature selection method to improve Backward Elimination performance for software defect prediction.
2. To further the research studying creation more optimal methods for software defects prediction.
3. Introducing additional methods to mitigate drawbacks of Backward Elimination.

II. MATERIAL AND METHOD

The proposed method of this research is, firstly determined relevant features from the dataset using correlation coefficient. Once all features have been ranked or weighted by their correlation coefficient value, selection of the top 20 and/ or top 10 relevant features based on their correlation coefficient value. This decision is based on the large number of different features in the NASA MDP dataset as show at TABLE . For datasets with 22 features, the top 10 (±50%) of the relevant features were selected, while datasets with 37-40 features will use the top 20 (±50%) and top 10 (±25%) relevant features. This standardization in feature selection is implemented for the stability and ease of research experimentation.

After selecting the features from the dataset, Backward Elimination was used to get the final model. Using Naïve Bayes as classification method to detect software defects and cross validation to evaluate the final model and to find the accuracy and Area under the curve (AUC). FIGURE 1 represents the research flow conducted in this study

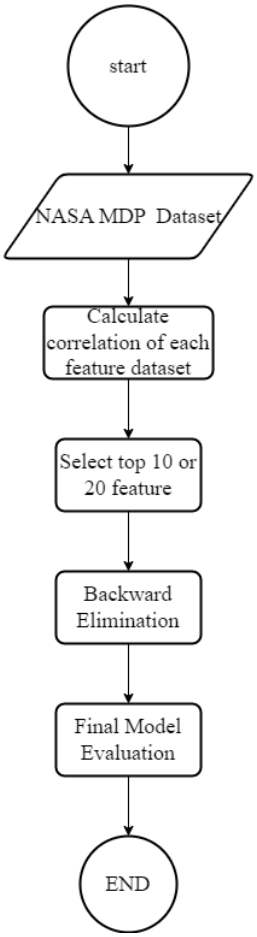

FIGURE 1. Research method Flowchart

A. DATASET

The NASA MDP dataset is a frequently used dataset by researchers conducting software defect prediction experiments [21], [22].Nasa MDP dataset is a collection of NASA Corpus which contains a diverse collection of real software projects spanning various domains and programming languages, including Java, C, and C++. This dataset features a wide range of code sizes, complexities, and functionalities, providing a comprehensive view of software development challenges. It includes numerous software metrics, such as lines of code, cyclomatic complexity, and other metric which offer valuable insights into the characteristics of software components. The primary aim of this dataset is to support the evaluation and development of predictive models designed to identify potentially defective software components early in the development process[23]. The Nasa MDP dataset used in this study was the cleaned version.

The criteria set in this study are dataset with a minimum sample size above 250, but not exceeding 2000 with defect percentages less than 13%, and a minimum of 20 defects. The dataset that meets these criteria are CM1, MC1, MW1, PC1, and PC3 as show in TABLE 1. These criteria are established to reduce required computational power, and overfitting which may occur if the sample data is too large, this is one of the detrimental characteristics of feature selection wrapper methods [6], [24], [25]. The Nasa MDP dataset has two class label Y and N. Class label Y is for defective attribute and class label N for non-defective attribute, in data preprocessing stage Label Y and N are converted to values of 1 and 0 respectively [26].

TABLE 1
NASA MDP DATASET

Dataset	Number of features	Sample size	Number of defects	Defect %
CM1	38	327	42	12.8
JM1	22	7720	1612	20.8
KC1	22	1162	294	25.3
KC3	40	194	36	18.5
MC1	38	1988	46	2.3
MC2	40	124	44	35.4
MW1	38	253	27	10.7
PC1	38	705	61	9.7
PC2	37	722	16	2.2
PC3	38	1077	134	12.4
PC4	38	1270	176	13.8
PC5	39	1694	458	27.0

B. FEATURE SELECTION

1. CORELLATION COEFFICIENT

Correlation coefficient is one of the simple filter algorithms, using a heuristic evaluation function to rank the feature subset based on their significance, which is determined by their high correlation. After identifying significant features from the dataset, these features are then ranked using various feature ranking techniques [27]. After the dataset is prepared, we calculate the correlation coefficient of each dataset feature to the target variable. Correlation Coefficient was used because it can address Naïve Bayes key weakness through determination of which features are relevant for software defect prediction.[15], [16].

It should be noted that though correlation coefficient is one method of determining feature relevancy, it has the risk because as some features may have high relevancy for the target variable, but doesn't mean it will good predictor of defect.

Nonetheless it a useful tool for defect prediction and can easily be implemented. Since we used correlation coefficient with Backward Elimination it should help mitigate the risk. Backward Elimination can help mitigated the risk by removing less significant feature, which may also reduce correlations among the remaining variables.

There are several methods to find the correlation coefficient of features, however as the dataset utilized contains binomial/dichotomous labels denoted as 'N' and 'Y' (or 0 and 1), equation (1) [28] is used to calculate the correlation feature of the dataset. TABLE 2 is the pseudocode to find the correlation coefficient of each feature (Eq. (1)).

$$r_{pb} = \frac{M_1 - M_0}{s_n} \sqrt{\frac{n_1 n_0}{n^2}} \quad (1)$$

M_1 denotes the mean value on the constant variable x for all data points in group 1, and M_0 denote the mean values of group 2. The dataset is grouped by their label, n_1 and n_0 which denotes the number of totals in group 1 and 2. n denotes the total of the sample size, and s_n is the standard deviation. Equation (2) [28] is used to calculate standard deviation (Eq. (20)).

$$s_n = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \quad (2)$$

After calculating the correlation coefficient of each feature, the result can be ranked. The closer the result is to zero the lower its rank, from this ranking we can choose how many features should be removed or selected for Backward Elimination. In this study we select the top 20 and top 10 feature.

TABLE 2
Correlation Coefficient pseudocode

Correlation Coefficients

Initialize:
Dataset with feature X (binary: Y or N) and target variable Y (continuous)
 	
Calculate the mean of Y as Y_mean
Calculate the standard deviation of Y as Y_std (Equation 2)
Calculate the mean of X as X_mean
Calculate the standard deviation of X as X_std (equation 2)
 	
Calculate the Point-Biserial Correlation coefficient (equation 1) for feature X and target variable Y:
For each data point in the dataset:
  Calculate the biserial correlation coefficient r_biserial as (mean of Y where X = 1 - mean of Y where X = 0) / Y_std
  Calculate the point-biserial correlation coefficient r_pb as r_biserial * sqrt(count(X=1) *count(X=0) / (count(X)-1))
 	
Return the average or final value of r_pb as the Point-Biserial Correlation coefficient for X and Y

2. BACKWARD ELIMINATION

Backward Elimination (BE) is a comparatively simpler variable/feature selection method. The main advantages of Backward Elimination are initial removal of the least important features and guaranteed retention of the most important features in the model. Some disadvantages include the removed features cannot be added back into the model which might improve the performance result [12]. Improving the performance of Backward Elimination for software defect prediction makes this method a promising option, as it produces favorable results and is simple to implement.

The next step after selecting the top 20 and/or top 10 features based on their correlation is to use Backward Elimination to find the final model of the dataset. Backward Elimination is a machine learning algorithm for feature selection from a dataset. The few advantages of Backward Elimination are improved performance and accuracy if not using any feature selection method, and decreasing complexity/ simplification of the model[29]. In general, Backward Elimination starts with a complete set of features, removing it's the least important variable feature repeatably until the desired parameter value is reached [8], [30].

The step of Backward Elimination involves the following:

1. Start with a Full Model: Begin with a model that includes all potential predictor variables.
2. Fit the Model: Use a statistical modeling technique to fit the full model to the data.
3. Identify the Least Significant Variable: Determine the variable with the highest p-value
4. Remove the Variable: If the variable identified in step 3 has a p-value above a specified significance level (e.g., 0.05), remove it from the model.
5. Refit the Model: Re-estimate the model without the removed variable. Repeat the Process: Continue steps 3-5 until all remaining variables in the model are statistically significant.
6. Repeat the Process: Continue steps 3-5 until all remaining variables in the model are statistically significant.

TABLE 3 is pseudocode of a typical Backward Elimination algorithm. The P-value or the parameter value can be set to anything, however usually it is set to 0.05. In this research the Backward Elimination stops when then next model has a decreasing performance result and/or stops after 10 features are removed. The stop condition can be changed, for example stopping if the accuracy result decrease by significant value of 0.03.

TABLE 3
Backward Elimination pseudocode

Backward Elimination

Initialize:
List of all features F = [f1, f2, ..., fn]
Selected features S = F
Significance level alpha
Model M
 	
while True:
  Train the model M using features in S
  Calculate the p-values for each feature in S
  Identify the feature with the largest p-value p_max
 	
  if p_max > alpha:
  Remove the feature with p_max from S
  else:

Break the loop as all remaining features are significant
Return: Selected features S

end

D. CLASSIFICATION

1. NAÏVE BAYES

Naïve Bayes (NB) is a classification technique based on Bayes' theory. It assumes all feature are independent to one another, this "naive" assumption hence why it is called Naïve Bayes [31]. It falls under the generative learning algorithm family, meaning it model input distribution for a given class or category. Naïve Bayes calculates the highest probability value and assigns the test data to the most appropriate category based on this result [31]. It is an efficient algorithm that achieves good accuracy with large dataset [2].

Naïve Bayes is considered a popular and effective method for supervised machine learning not only due to its simplicity, but also being high performance and robust [31], [32]. Moreover, leveraging feature selection with Naïve Bayes can significantly enhance its performance[3]. It is simple to implement, and often yields better performance than comparatively more complex classification methods[1], [33]. However Naïve Bayes assumes that all features are equally important, which may not always be accurate as different features serve different purposes. Additionally, not all are reliable indicators of software defects[1]. An alternative to address this uses weighting if the features based on their importance, this approach known as weighted Naïve Bayes (WNB). [34], [35].

Naïve Bayes is a frequently used classification method in software defect prediction. It used because of its high prediction efficiency and potential to handle imbalanced datasets[2], [22], [36]. Eq. (3) [2] is Naïve Bayes algorithm for prediction/classification in a dataset.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \quad (3)$$

In this equation D represents data with an unknown class, while H denote the hypothesis that D belongs to a specific class. P(h) denote the prior probability, while P(D|h) denotes the posterior probability, providing the probability of the data given the hypothesis. Additionally, P(D) denotes the probability of the data or predictors prior probability. TABLE 4 is the Naïve Bayes algorithm in pseudocode.

TABLE 4
Naïve Bayes algorithm pseudocode

Naïve Bayes algorithm

Start

Input training dataset

Separate the dataset by class labels

Calculate the prior probability of each class label

For each feature in the dataset:
a. Calculate the likelihood of the feature given each class label
b. Calculate the conditional probability of each class label given the feature

Input test dataset
For each instance in the test dataset:
a. Calculate the posterior probability of each class label given the instance
b. Predict the class label with the highest posterior probability
Output the predicted class labels for the test dataset

End

E. MODEL EVALUATION

1. K-FOLD CROSS VALIDATION

After selecting relevant features and applying Backward Elimination to find the final model, the performance of the final model can be assessed using cross validation. Cross validation is a testing system for machine learning algorithm [37]. Cross-validation is a technique used to assess and improve the performance of a model while reducing systematic errors. It works by dividing the data into multiple subsets. In k-fold cross-validation, the dataset is split into k equal-sized folds (e.g., k=10). Each fold serves as a test set once, while the remaining k−1 folds are combined to form the training set. This process is repeated k times, with each fold being used as the test set exactly once. By averaging the results from these multiple iterations, cross-validation provides a more reliable estimate of the model's performance and helps to prevent overfitting. In this study, we will use 10-fold cross-validation. The data will be divided into ten subsets, with each subset containing instances from the same class. [23], [38].

2. AREA UNDER THE ROC CURVE

After the final model is tested by cross validation, equation (4) and equation (5) [4] are used to check the model accuracy and Area under the curve (AUC). Accuracy represents the proportion of correctly predictive data by the model on the test set, covering both positive and negative result. several researchers have corroborated that accuracy can be insufficient and even a misleading performance metric when dealing with imbalanced datasets. In our case, the dataset is imbalance. To handle this imbalance, we have used AUC, a metric used to evaluate the performance of classification models [39]. AUC is a metric used to evaluate the performance of classification models [40]. It quantifies the model's ability to distinguish between positive and negative classes, with values ranging from 0 to 1. AUC measures the separability between the true positive rate and the false positive rate across different threshold settings. Essentially, a higher AUC indicates better model performance in distinguishing between classes [23].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

$$AUC = \frac{1 + TPR - FPR}{2} \quad (5)$$

TP is true positive where positive cases correctly classify as positive, FP is false positive where negative cases incorrectly classify as positive, TN is true negative where negative cases correctly classify as negative, and FN is false negative where

positive cases incorrectly classify as negative. To find this information, matrix confusion is used. Equation (6) and equation (7) [41]used to find TPR and FPR, TPR is true positive rate and FPR is false positive rate[41].

$$TPR = \frac{TP}{TP + FN} \quad (6)$$

$$FPR = \frac{FP}{FP + TN} \quad (7)$$

TABLE 5 is the pseudocode step for final model evaluation.

TABLE 5
Model Evaluation pseudocode

Model Evaluation

```
Start
# Function to calculate True Positive (TP), True Negative (TN), False
Positive (FP), False Negative (FN)

def calculate_confusion_matrix(actual_values, predicted_values):
    TP = 0
    TN = 0
    FP = 0
    FN = 0

    for i in range(len(actual_values)):
        if actual_values[i] == 1 and predicted_values[i] == 1:
            TP += 1
        elif actual_values[i] == 0 and predicted_values[i] == 0:
            TN += 1
        elif actual_values[i] == 0 and predicted_values[i] == 1:
            FP += 1
        else:
            FN += 1

    return TP, TN, FP, FN

# Function to calculate Accuracy / (Equation (4)
def calculate_accuracy(TP, TN, FP, FN):
    accuracy = (TP + TN) / (TP + TN + FP + FN)
    return accuracy

# Function to calculate AUC
def calculate (actual_values, predicted_values):
    # Calculate True Positive Rate (TPR) / Equation (6) and False
Positive Rate (FPR) / Equation (7)

    for threshold in range (0, 101):
        threshold = threshold / 100.0
        TP, TN, FP, FN = calculate_confusion_matrix(actual_values, [1
if val >= threshold else 0 for val in predicted_values])

        tpr = TP / (TP + FN)
        fpr = FP / (FP + TN)

        tpr_list.append(tpr)
        fpr_list.append(fpr)

    # Calculate AUC/Equation (5)
AUC = 0
AUC = (1+TPR-FPR)/2

    return auc
```

III. RESULTS

This section shows result after using cross validation with Naïve Bayes as prediction algorithm to find the result of accuracy and Area Under the Curve (AUC) on the final model from each data set. Results are divided into several tables that shows the accuracy result and AUC result of Naïve Bayes only, Backward Elimination only, top 20 feature based on its correlation coefficient with Backward Elimination, and top 10 feature based on its correlation coefficient feature with Backward Elimination.

TABLE 6
Naïve Bayes Accuracy Result

Dataset	Naïve Bayes
CM1	81%
MC1	90%
MW1	79%
PC1	88%
PC3	33%
Average	**74.2%**

TABLE 6 is the accuracy result of Naïve Bayes classification without selecting feature with correlation or using Backward Elimination. From the table the results are follows: CM1: 81%, MC1: 90%, MW1: 79%, PC1: 88% and PC3: 33%. MC1 had the highest accuracy and PC3 had the lowest accuracy. The average accuracy is 74.2%. Overall using Naïve Bayes for software defect prediction yielded good results, however from the table we can see the results for PC3 was poor, hence why for software defect prediction feature selection is needed.

TABLE 1
Backward Elimination Accuracy Result

Dataset	Backward Elimination
CM1	81%
MC1	91%
MW1	79%
PC1	87%
PC3	30%
Average	**73.6%**

TABLE 7 is the accuracy result of Backward Elimination with Naïve Bayes classification as its prediction algorithm to dataset. From the table the results are: CM1: 81%, MC1: 91%, MW1: 79%, PC1: 87%, and PC3: 30%. The highest result was MC1, the lowest was PC3, and with the average result was 73.6% accuracy. There were some improvements for one dataset however this was minimal, and there were 2 datasets showing decreased accuracy making the average result poorer than only using Naïve Bayes. This means using Backward Elimination only is unideal accurate software defect prediction as it only improved one of dataset accuracy out of the five tested.

TABLE 2
Backward Elimination with top 20 Accuracy Result

Dataset	Top 20 with Backward Elimination
CM1	82%
MC1	92%
MW1	81%
PC1	89%
PC3	76%

Average	**84%**

TABLE 8 is the accuracy result of selecting top 20 features based on their correlation with Backward Elimination to further select from the top 20 selected feature and using Naïve Bayes as its prediction algorithm. From the table, the result is as follows: CM1: 82%, MC1: 92%, MW1: 81%, PC1: 89%, and PC3: 76%, with the highest result from MC1, the lowest from PC3 with and an average result of 84% accuracy. It is apparent from the result table we can see a substantial improvement for PC3 accuracy using top 20 with Backward Elimination, and other datasets also increased accuracy also increased slightly, resulting in a higher average result compare to the other two methods tested.

TABLE 9
Backward Elimination with top 10 Accuracy Result

Dataset	Top 10 with Backward Elimination
CM1	83%
MC1	92%
MW1	83%
PC1	88%
PC3	84%
Average	**86%**

TABLE 9 is the accuracy result of selecting the top 10 features based on their correlation with Backward Elimination to further select from the top 10 selected features, using Naïve Bayes as its prediction algorithm. From the table the result is as follows: CM1: 83%, MC1: 92%, MW1: 83%, PC1: 88%, and PC3: 84% with the highest result being MC1, the lowest being CM1 and an average result of 86% accuracy.

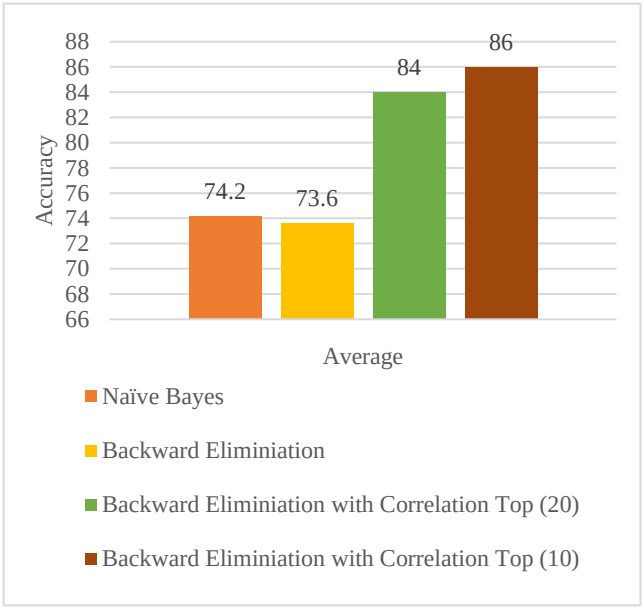

FIGURE 2. Average Accuracy Comparison

As show in FIGURE 2 and FIGURE 3 the accuracy result that this far shown from each dataset used in this study, it is evident from testing that top 10 with Backward Elimination yielded the best result of accuracy from methods tested as indicated by higher average accuracy, and also individual accuracy results. This mean top 10 with Backward Elimination has better accuracy performance when compare to other methods used in this study.

TABLE 10
Naïve Bayes AUC Result

Dataset	Naïve Bayes
CM1	0.688
MC1	0.7
MW1	0.556
PC1	0.802
PC3	0.762
Average	**0.702**

TABLE 10 is the AUC result of each dataset using Naïve Bayes classification. Naïve Bayes AUC yielded the following results: CM1: 0.688, MC1: 0.7, MW1: 0.556, PC1: 0.802, PC3: 0.762 with an average result of 0.702. The highest AUC was PC1, and the lowest AUC result was MW1.

TABLE 11
Backward Elimination AUC Result

Dataset	Backward Elimination
CM1	0.756
MC1	0.774
MW1	0.791
PC1	0.81
PC3	0.793
Average	**0.785**

TABLE 11 is the AUC result of each dataset using Backward Elimination and Naïve Bayes classification. Backward Elimination AUC yielded the following results: CM1: 0.756, MC1: 0.774, MW1: 0.791, PC1: 0.81, PC3: 0.793 with an average result of 0.785. The highest AUC was

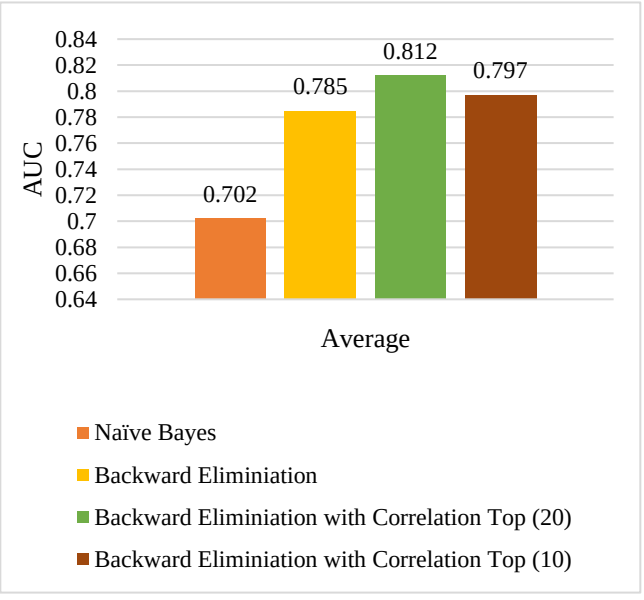

FIGURE 3. Average AUC Comparison

PC1 and the lowest AUC result was CM1. From the result it shows that Backward Elimination not only improved each

dataset AUC result, but its overall result compared to when only using Naïve Bayes.

TABLE 12
Backward Elimination with top 20 AUC Result

Dataset	Top 20 with Backward Elimination
CM1	0.786
MC1	0.815
MW1	0.824
PC1	0.813
PC3	0.821
Average	**0.812**

TABLE 12 is the AUC result of each dataset using Backward Elimination with the top 20 features and Naïve Bayes classification. Backward Elimination with top 20 AUC yielded the following results: CM1: 0.786, MC1: 0.815, MW1: 0.824, PC1: 0.813, PC3: 0.821 with the average result 0.812. The highest AUC was MW1 and the lowest AUC result was CM1. It is apparent that selecting top 20 with Backward Elimination yields improved individual and overalls results, the higher AUC result.

TABLE 13
Backward Elimination with top 10 AUC Result

Dataset	Top 10 with Backward Elimination
CM1	0.767
MC1	0.826
MW1	0.8
PC1	0.777
PC3	0.814
Average	**0.797**

TABLE 13 is the AUC result of each dataset using the top 10 features with Backward Elimination and Naïve Bayes classification. Backward Elimination with top 10 AUC yielded the following results: CM1: 0.767, MC1: 0.826, MW1: 0.8, PC1: 0.777, PC3: 0.814 with an average result of 0.797. The highest AUC was MC1 and the lowest AUC result was PC1. As show in FIGURE 3 and FIGURE 5, when comparing against previous AUC results, it is apparent top 10 with Backward Elimination yielded better individual and average result than Backward Elimination alone, however poorer result compared to top 20 with Backward Elimination this variation suggests the number feature that were selected prior to applying Backward Elimination. Specially, selecting a small subset of top feature (e.g., the top 10) may not capture sufficient relevant information, leading to less optimal results. In contrast, incorporating a larger number of features (e.g., the top 20) before applying Backward Elimination appears to provide a more comprehensive representation of the dataset, thereby improving the AUC performance.

Therefore, the results highlight that the effectiveness of Backward Elimination can be highly dependent on the number of features initially selected. A larger feature set before applying Backward Elimination generally yields better performance, suggesting that more features can help in capturing the relevant patterns and improving model accuracy. This finding underscores the importance of carefully considering the feature selection process and the number of features used in conjunction with Backward Elimination to optimize performance.

TABLE 14 to TABLE 17 display the confusion matrix results for each method used in this study. The rows labeled "Pred N" and "Pred Y" represent the number of times the model predicted "N" or "Y," respectively. The columns labeled "True N" and "True Y" show the actual number of instances where the true class was "N" or "Y". In these tables, the number at the intersection of the "Pred N" row and the "True N" column indicates how many times the method correctly predicted "N" when the true class was also "N". Conversely, the number at the intersection of the "Pred N" row and the "True Y" column shows how many times the method predicted "N" when the true class was actually "Y", indicating a misclassification.

TABLE 14
Naïve Bayes Confusion Matrix Result

Dataset		True N	True Y
CM1	Pred N	255	29
	Pred Y	30	13
		True N	True Y
MC1	Pred N	1780	32
	Pred Y	162	14
		True N	True Y
MW1	Pred N	187	12
	Pred Y	39	15
		True N	True Y
PC1	Pred N	603	39
	Pred Y	41	22
		True N	True Y
PC3	Pred N	238	13
	Pred Y	705	121

TABLE 14 is the Confusion Matrix Result of Naïve Bayes yielded the following result: CM1: 268 correct predictions and incorrect predictions 59, MC1: 1794 correct predictions and 194 incorrect predictions, MW1: 202 correct predictions and 51 incorrect predictions, PC1: 625 correct predictions and 80 incorrect predictions, PC3: 359 correct predictions and 715 incorrect predictions. Overall, using Naïve Bayes only yield a decent result except for PC3 which is reflected to the result on TABLE 6.

TABLE 15
Backward Elimination Confusion Matrix Result

Dataset		True N	True Y
CM1	Pred N	253	30
	Pred Y	32	12
		True N	True Y
MC1	Pred N	1803	33
	Pred Y	139	13
		True N	True Y
MW1	Pred N	185	11
	Pred Y	41	16
		True N	True Y
PC1	Pred N	598	39
	Pred Y	46	22

		True N	True Y
PC3	Pred N	204	8
	Pred Y	739	126

TABLE 15 is the Confusion Matrix Result of Naïve Bayes yielded the following result: CM1: 265 correct predictions and incorrect predictions 62, MC1: 1816 correct predictions and 172 incorrect predictions, MW1: 201 correct predictions and 52 incorrect predictions, PC1: 620 correct predictions and 85 incorrect predictions, PC3: 330 correct predictions and 747 incorrect predictions. Compared to Table 14 results, there is minimal improvement in performance. In particular, the performance for PC3 has deteriorated, as indicated by the significantly higher number of incorrect predictions. This decline in performance is further reflected in the average accuracy results presented in TABLE 7, suggesting a general decrease in the classifier's effectiveness.

TABLE 16
Backward Elimination with top 20 Confusion Matrix Result

Dataset		True N	True Y
CM1	Pred N	253	28
	Pred Y	32	14
		True N	True Y
MC1	Pred N	1824	36
	Pred Y	118	10
		True N	True Y
MW1	Pred N	192	12
	Pred Y	34	15
		True N	True Y
PC1	Pred N	608	39
	Pred Y	36	22
		True N	True Y
PC3	Pred N	737	43
	Pred Y	206	91

TABLE 16 is the Confusion Matrix Result of Naïve Bayes yielded the following result: CM1: 267 correct predictions and incorrect predictions 60, MC1: 1834 correct predictions and 154 incorrect predictions, MW1: 207 correct predictions and 46 incorrect predictions, PC1: 630 correct predictions and 75 incorrect predictions, PC3: 829 correct predictions and 249 incorrect predictions. Compare to Table 15 result, there are several improvements, notably the number of correct predictions for PC3 increase from 330 to 829 correct prediction this result also reflected at TABLE 8, indicating a positive trend in the performance.

TABLE 17
Backward Elimination with top 10 Confusion Matrix Result

Dataset		True N	True Y
CM1	Pred N	261	29
	Pred Y	24	13
		True N	True Y
MC1	Pred N	1830	35
	Pred Y	112	11
		True N	True Y
MW1	Pred N	196	12
	Pred Y	30	15
		True N	True Y
PC1	Pred N	605	39
	Pred Y	39	22
		True N	True Y
PC3	Pred N	859	88
	Pred Y	84	46

TABLE 17 is the Confusion Matrix Result of Naïve Bayes yielded the following result: CM1: 274 correct predictions and incorrect predictions 53, MC1: 1841 correct predictions and 147 incorrect predictions, MW1: 211 correct predictions and 42 incorrect predictions, PC1: 627 correct predictions and 85 incorrect predictions, PC3: 905 correct predictions and 172 incorrect predictions. Compare to TABLE 16, these results show an overall improvement which also reflected to TABLE 9 result.

Based on the results presented in TABLE 6 to 13, we can make several observations. TABLE 7 shows that Backward Elimination yields only 30% accuracy for the PC3 dataset, indicating that it is not an effective method for this dataset. In contrast, TABLES 8 and 9 demonstrate that the proposed method achieves accuracies of 76% and 84%, respectively, suggesting that using correlation coefficient feature selection improves the performance of Backward Elimination and this suggest that the performance of Backward Elimination can be affected if the feature didn't get remove. This result highlighted that Backward Elimination may not be sufficient and could benefit from combining with other method to select better feature for the final model.

Overall, the proposed method outperforms both Backward Elimination and the Naïve Bayes which are the baseline model. For a visual representation results comparison of the baseline model to the proposed method, refer to FIGURE 2,3,4 and 5, which graphically depict the data from TABLE 6 to 13. FIGURE 2 and 3 show the average results for each method, while FIGURE 4 and 5 present a detailed comparison of results across different datasets as well as average results for each method.

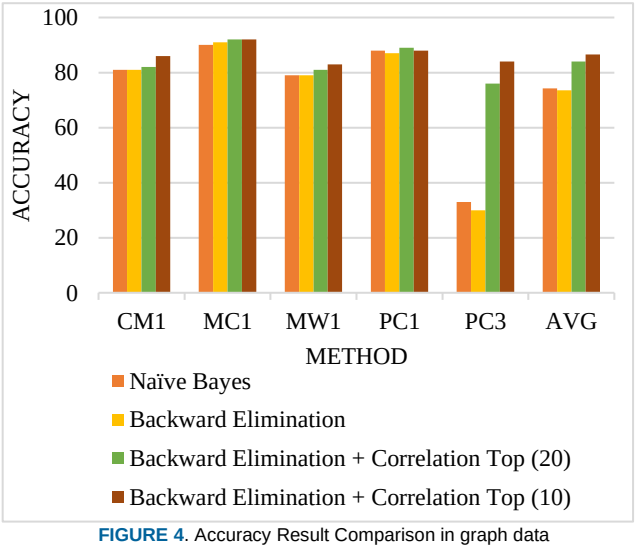

FIGURE 4. Accuracy Result Comparison in graph data

IV. DISSCUSION

In this study we evaluated the effectiveness of selecting relevant features based on the correlation coefficient of each feature to the class feature, then using Backward Elimination

to further select the relevant features. This was expected to mitigate the drawbacks of Backward Elimination, improving Backward Elimination performance for software defect prediction. FIGURE 4 shows graphed accuracy result. and FIGURE 5 shows graphed AUC result for easier visualization and comparison of each methods results offering a clear overview of the performance metrics. FIGURE 4 shows almost the same Improvement as FIGURE 5, except when using the top 10 features with Backward Elimination the average result of the accuracy is better than using the top 20 with Backward Elimination. This suggests that the features that were remove in test with top 10 and Backward Elimination was better for the accuracy result but comparatively worse for the AUC result.

FIGURE 5 shows that using Backward Elimination does improve the average AUC result compared to using all feature in the dataset. Additionally, selecting the top 20 or top 10 relevant features based on their correlation coefficient before applying Backward Elimination improved the average AUC result compared to only using Backward Elimination and Naïve Bayes Only.

The average results of AUC from testing the top 20 features and Backward Elimination were better that using the top 10 features and Backward Elimination. This suggests that some of the features that were removed during the process selecting top 10 relevant feature play a role in enhancing the AUC result when they are included in the final model.

From both result its evident that the features that are selected affect different results either it affects accuracy or AUC depend by the features that were selected, meaning selecting different subset feature before using Backward Elimination can have better or worse performance result. Despite this, using correlation coefficient feature selection with Backward Elimination can improve the performance of accuracy and AUC by a substantial amount around ±40% as seen in FIGURE 4 PC3.

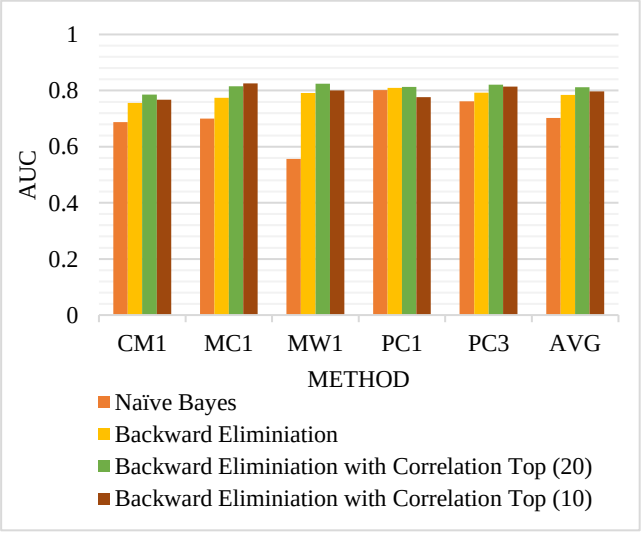

FIGURE 5. AUC Result Comparison in graph data

Another thing to note is that the result of accuracy and AUC can be affected by the classification method used. Different classification methods have their own weaknesses and advantages that may affect the performance results or overall model performance.

TABLE 18
AUC Result comparison of the Proposed Method with Other Studies

Method		BE with top 20	Enchaced WFS [22]	RAHFMWS [17]	HFS BE [30]
Dataset	CM1	0.786	0.722	0.744	0.746
	MW1	0.824	0.756	0.782	0.92
	PC1	0.784	0.826	0.792	0.795
	PC3	0.821	0.806	0.806	0.782
Average		**0.804**	**0.781**	**0.778**	**0.811**

TABLE 18 show how the propose method compare to the other method. The proposed method demonstrates improvements over other two previous study method. Although the proposed method did not yield overall results compared to HFS_BE, the proposed method still yields better result at CM1 and PC3 dataset. Our study did not explore other classification algorithms or optimize correlation feature selection. Although HFS_BE currently seems the best among the methods compared, its effectiveness for software defect prediction is still unclear based on the CM1 and PC3 results.

there are limitations in this study and for the propose method. the limitations are as follow: our dependence on specific datasets limits the generalizability of our findings. Additionally, by focusing on the AUC metric and one classification algorithms may overlook other important aspects of evaluating model performance, our method does not explore the potential benefits of reintroducing removed features into the final model, which could potentially improve performance. The results can also vary based on the number of features selected before applying Backward Elimination, and we did not consider the impact of removing top relevant features, which might affect performance.

Despite these limitations, our study demonstrates that Backward Elimination has potential for software defect prediction, particularly when used with other methods. We also show that correlation coefficient filter in conjunction with other techniques, can enhance performance. Further research is needed to determine the optimal methods for integrating Backward Elimination to improve its effectiveness in software defect prediction. This exploration could lead to the development of more accurate and robust prediction models.

V. CONCLUSION

The aim of this study was to determine if selecting several features that were considered relevant before applying Backward Elimination can mitigate the drawbacks of Backward Elimination whilst also improving the prediction performance of the model. Hence, correlation coefficient is used to determine which features are relevant or important for software defect prediction. By choosing several features

before applying Backward Elimination, improvement of the performance of Backward Elimination was observe.

The performance of the final model was evaluated with cross validation using performance metric of accuracy and AUC from the software defect prediction result. After testing, it was found that the average result of selecting some relevant features before using Backward Elimination outperformed that only using Backward Elimination based on average AUC and average accuracy results. Backward Elimination had average AUC and accuracy results of 0.785 AUC, and 73.6% accuracy. The average result for top 10 features with Backward Elimination was 0,797 AUC, and 86.6% accuracy. The average result of top 20 feature with Backward Elimination was 0.812 AUC and 84% accuracy. While the performance result may vary depending on the selected features, overall, from the result it appears using correlation coefficient to determine the relevant features does improve the performance of the Backward Elimination, helping mitigate the drawbacks of Backward Elimination.

For future studies, areas for further study include testing of different methods for relevant features selection such as weighted by Gain Ratio. Utilizing different method may minorly, or significantly change the top relevant features, yielding different performance results. Investigation of this to determine the optimal feature selection method is expected to be beneficial for achieving further improvement in defect detection. Another area for consideration is utilization of different classification methods like K-Nearest Neighbors (KNN). usage of different classification method, is also expected to prioritize different sets of features bring a new set of weaknesses and advantages which may be more suitable for defect detection. By using different methods to find relevant features, and different classification methods, we hope that further research can find the best method to pair with Backward Elimination, improving its performance and mitigating its drawback for better software defect prediction.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to the individual from the Computer Sciences department at Lambung Mangkurat University for their invaluable contribution, their input and suggestion greatly enhance the quality of our work. We also appreciate hard work and cooperation of our project team member.

REFERENCES

[1] H. Ji, S. Huang, Y. Wu, Z. Hui, and C. Zheng, "A new weighted naive Bayes method based on information diffusion for software defect prediction," *Software Quality Journal*, vol. 27, no. 3, pp. 923–968, Sep. 2019, doi: 10.1007/s11219-018-9436-4.

[2] A. Rahim, Z. Hayat, M. Abbas, A. Rahim, and M. A. Rahim, "Software Defect Prediction with Naïve Bayes Classifier," in *Proceedings of 18th International Bhurban Conference on Applied Sciences and Technologies, IBCAST 2021*, Institute of Electrical and Electronics Engineers Inc., Jan. 2021, pp. 293–297. doi: 10.1109/IBCAST51254.2021.9393250.

[3] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques," *International Journal of Applied Science and Engineering*, vol. 17, no. 4, pp. 331–344, Jan. 2020, doi: 10.6703/IJASE.202012_17(4).331.

[4] A. Iqbal *et al.*, "Performance analysis of machine learning techniques on software defect prediction using NASA datasets," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 5, pp. 300–308, 2019, doi: 10.14569/ijacsa.2019.0100538.

[5] T. M. P. Hà, T. M. H. Le, and T. B. Nguyen, "A Comparative Analysis of Filter-Based Feature Selection Methods for Software Fault Prediction," *Journal of Research and Development on Information and Communication Technology*, pp. 1–7, Jun. 2021, doi: 10.32913/mic-ict-research-vn.v2021.n1.969.

[6] U. M. Khaire and R. Dhanalakshmi, "Stability of feature selection algorithm: A review," Apr. 01, 2022, *King Saud bin Abdulaziz University*. doi: 10.1016/j.jksuci.2019.06.012.

[7] D. Singh and B. Singh, "Investigating the impact of data normalization on classification performance," *Appl Soft Comput*, vol. 97, Dec. 2020, doi: 10.1016/j.asoc.2019.105524.

[8] B. Venkatesh and J. Anuradha, "A review of Feature Selection and its methods," *Cybernetics and Information Technologies*, vol. 19, no. 1, pp. 3–26, 2019, doi: 10.2478/CAIT-2019-0001.

[9] R. K. Agrawal, B. Kaur, and S. Sharma, "Quantum based Whale Optimization Algorithm for wrapper feature selection," *Applied Soft Computing Journal*, vol. 89, Apr. 2020, doi: 10.1016/j.asoc.2020.106092.

[10] D. Singh and B. Singh, "Investigating the impact of data normalization on classification performance," *Appl Soft Comput*, vol. 97, Dec. 2020, doi: 10.1016/j.asoc.2019.105524.

[11] W. Sauerbrei *et al.*, "State of the art in selection of variables and functional forms in multivariable analysis—outstanding issues," *Diagn Progn Res*, vol. 4, no. 1, p. 3, Dec. 2020, doi: 10.1186/s41512-020-00074-3.

[12] M. Z. I. Chowdhury and T. C. Turin, "Variable selection strategies and its importance in clinical prediction modelling," *Fam Med Community Health*, vol. 8, no. 1, Feb. 2020, doi: 10.1136/fmch-2019-000262.

[13] J. R. Busenbark, H. Yoon, D. L. Gamache, and M. C. Withers, "Omitted Variable Bias: Examining Management Research With the Impact Threshold of a Confounding Variable (ITCV)," *J Manage*, vol. 48, no. 1, pp. 17–48, Jan. 2022, doi: 10.1177/01492063211006458.

[14] L. Jiang, L. Zhang, C. Li, and J. Wu, "A Correlation-Based Feature Weighting Filter for Naive Bayes," *IEEE Trans Knowl Data Eng*, vol. 31, no. 2, pp. 201–213, Feb. 2019, doi: 10.1109/TKDE.2018.2836440.

[15] M. Kondo, C. P. Bezemer, Y. Kamei, A. E. Hassan, and O. Mizuno, "The impact of feature reduction techniques on defect prediction models," *Empir Softw Eng*, vol. 24, no. 4, pp. 1925–1963, Aug. 2019, doi: 10.1007/s10664-018-9679-5.

[16] S. Ruan, B. Chen, K. Song, and H. Li, "Weighted naïve Bayes text classification algorithm based on improved distance correlation coefficient," *Neural Comput Appl*, vol. 34, no. 4, pp. 2729–2738, Feb. 2022, doi: 10.1007/s00521-021-05989-6.

[17] A. O. Balogun *et al.*, "A novel rank aggregation-based hybrid multifilter wrapper feature selection method in software defect prediction," *Comput Intell Neurosci*, vol. 2021, 2021, doi: 10.1155/2021/5069016.

[18] J. Zhang, Y. Xiong, and S. Min, "A new hybrid filter/wrapper algorithm for feature selection in classification," *Anal Chim Acta*, vol. 1080, pp. 43–54, Nov. 2019, doi: 10.1016/j.aca.2019.06.054.

[19] M. Shantal, Z. Othman, and A. A. Bakar, "A Novel Approach for Data Feature Weighting Using Correlation Coefficients and Min–Max Normalization," *Symmetry (Basel)*, vol. 15, no. 12, Dec. 2023, doi: 10.3390/sym15122185.

[20] M. N. M. Rahman, R. A. Nugroho, M. R. Faisal, F. Abadi, and R. Herteno, "Optimized multi correlation-based feature selection in software defect prediction," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 22, no. 3, pp. 598–605, Jun. 2024, doi: 10.12928/TELKOMNIKA.v22i3.25793.

[21] Y. Liu, W. Zhang, G. Qin, and J. Zhao, "A comparative study on the effect of data imbalance on software defect prediction," in *Procedia Computer Science*, Elsevier B.V., 2022, pp. 1603–1616. doi: 10.1016/j.procs.2022.11.349.

[22] A. O. Balogun *et al.*, "Software defect prediction using wrapper feature selection based on dynamic re-reranking strategy," *Symmetry (Basel)*, vol. 13, no. 11, Nov. 2021, doi: 10.3390/sym13112166.

[23] Angga Maulana Akbar, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, "Optimizing Software Defect Prediction Models: Integrating Hybrid Grey Wolf and Particle Swarm Optimization for Enhanced Feature Selection with Popular Gradient Boosting

Algorithm," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 2, pp. 169–181, Apr. 2024, doi: 10.35882/jeeemi.v6i2.388.
[24] H. Liu, M. Zhou, and Q. Liu, "An embedded feature selection method for imbalanced data classification," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 3, pp. 703–715, May 2019, doi: 10.1109/JAS.2019.1911447.
[25] X. Ying, "An Overview of Overfitting and its Solutions," in *Journal of Physics: Conference Series*, Institute of Physics Publishing, Mar. 2019. doi: 10.1088/1742-6596/1168/2/022022.
[26] M. K. Suryadi, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, "A Comparative Study of Various Hyperparameter Tuning on Random Forest Classification with SMOTE and Feature Selection Using Genetic Algorithm in Software Defect Prediction," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 2, pp. 137–147, Apr. 2024, doi: 10.35882/jeeemi.v6i2.375.
[27] C. B. Gokulnath and S. P. Shantharajah, "An optimized feature selection based on genetic approach and support vector machine for heart disease," *Cluster Comput*, vol. 22, pp. 14777–14787, Nov. 2019, doi: 10.1007/s10586-018-2416-4.
[28] D. G. Bonett, "Point-biserial correlation: Interval estimation, hypothesis testing, meta-analysis, and sample size determination," *British Journal of Mathematical and Statistical Psychology*, vol. 73, no. S1, pp. 113–144, Nov. 2020, doi: 10.1111/bmsp.12189.
[29] F. Maulidina, Z. Rustam, S. Hartini, V. V. P. Wibowo, I. Wirasati, and W. Sadewo, "Feature optimization using Backward Elimination and Support Vector Machines (SVM) algorithm for diabetes classification," in *Journal of Physics: Conference Series*, IOP Publishing Ltd, Mar. 2021. doi: 10.1088/1742-6596/1821/1/012006.
[30] Y. Jian, X. Yu, Z. Xu, and Z. Ma, "A hybrid feature selection method for software fault prediction," *IEICE Trans Inf Syst*, vol. E102D, no. 10, pp. 1966–1975, 2019, doi: 10.1587/transinf.2019EDP7033.
[31] Putri Nabella, Rudy Herteno, Setyo Wahyu Saputro, Mohammad Reza Faisal, and Friska Abadi, "Impact of a Synthetic Data Vault for Imbalanced Class in Cross-Project Defect Prediction," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 2, pp. 219–230, Apr. 2024, doi: 10.35882/jeeemi.v6i2.409.
[32] I. Wickramasinghe and H. Kalutarage, "Naive Bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation," *Soft comput*, vol. 25, no. 3, pp. 2277–2293, Feb. 2021, doi: 10.1007/s00500-020-05297-6.
[33] L. Jiang, L. Zhang, L. Yu, and D. Wang, "Class-specific attribute weighted naive Bayes," *Pattern Recognit*, vol. 88, pp. 321–330, Apr. 2019, doi: 10.1016/j.patcog.2018.11.032.
[34] S. Chen, G. I. Webb, L. Liu, and X. Ma, "A novel selective naïve Bayes algorithm," *Knowl Based Syst*, vol. 192, Mar. 2020, doi: 10.1016/j.knosys.2019.105361.
[35] H. Zhang, L. Jiang, and L. Yu, "Attribute and instance weighted naive Bayes," *Pattern Recognit*, vol. 111, Mar. 2021, doi: 10.1016/j.patcog.2020.107674.
[36] A. O. Balogun *et al.*, "Impact of feature selection methods on the predictive performance of software defect prediction models: An extensive empirical study," *Symmetry (Basel)*, vol. 12, no. 7, Jul. 2020, doi: 10.3390/sym12071147.
[37] O. Karal, "Performance comparison of different kernel functions in SVM for different k value in k-fold cross-validation," in *Proceedings - 2020 Innovations in Intelligent Systems and Applications Conference, ASYU 2020*, Institute of Electrical and Electronics Engineers Inc., Oct. 2020. doi: 10.1109/ASYU50717.2020.9259880.
[38] Y. F. Zamzam, T. H. Saragih, R. Herteno, Muliadi, D. T. Nugrahadi, and P. H. Huynh, "Comparison of CatBoost and Random Forest Methods for Lung Cancer Classification using Hyperparameter Tuning Bayesian Optimization-based," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 2, pp. 125–136, Apr. 2024, doi: 10.35882/jeeemi.v6i2.382.
[39] R. Malhotra, R. Kapoor, P. Saxena, and P. Sharma, "SAGA: A Hybrid Technique to handle Imbalance Data in Software Defect Prediction," in *ISCAIE 2021 - IEEE 11th Symposium on Computer Applications and Industrial Electronics*, Institute of Electrical and Electronics Engineers Inc., Apr. 2021, pp. 331–336. doi: 10.1109/ISCAIE51753.2021.9431842.
[40] D. Valero-Carreras, J. Alcaraz, and M. Landete, "Comparing two SVM models through different metrics based on the confusion matrix," *Comput Oper Res*, vol. 152, Apr. 2023, doi: 10.1016/j.cor.2022.106131.
[41] C. S. Hong and T. G. Oh, "TPR-TNR plot for confusion matrix," *Commun Stat Appl Methods*, vol. 28, no. 2, pp. 161–169, 2021, doi: 10.29220/CSAM.2021.28.2.161.

BIBLIOGRAPHY

Muhammad Noor was born in Central Kalimantan, Indonesia, after finishing high school, he decided to pursue further study in Computer Science. Since 2020, he has been pursuing his studies at Universitas Lambung Mangkurat, where his research primarily focuses on software engineering. His academic work is concentrated on predicting software defects, an area he explored in-depth for his final project. he aims to contribute to advancements in software reliability through his research, striving to enhance the accuracy and effectiveness of software defect prediction methodologies in software development. His ongoing work reflects a strong commitment to improving software quality and engineering practices

Radityo Adi Nugroho holds a bachelor's degree in Informatics from the Islamic University of Indonesia and a master's degree in Computer Science from Gadjah Mada University. Currently, he is an assistant professor in the Department of Computer Science at Lambung Mangkurat University. His research interests include software defect prediction and computer vision. In addition to his academic role, he is also an IT practitioner with substantial experience, serving as a project manager and systems analyst. He has extensive experience in developing software and information systems for universities. Radityo Adi Nugroho is available for consultation and can be contacted at raditYo.adi@ulm.ac.id.

Setyo Wahyu Saputro is a lecturer in Computer Science Department, Faculty of Mathematics and Natural Science, Lambung Mangkurat University in Banjarbaru. He received bachelor's degree also in Computer Science from Lambung Mangkurat University in 2011, and received his master's degree in Informatics from STMIK Amikom University in 2016. He is active as an information technology practitioner and consultant, being a project manager or systems analyst working on several projects in government and private agencies in South Kalimantan province since 2017. His research interests include software engineering, human computer interaction, and artificial intelligence applications. He can be contacted at email: setyo.saputro@ulm.ac.id

Rudy Herteno, was born in Banjarmasin, South Kalimantan. After graduating from high school, he pursued his undergraduate studies in the Computer Science Department at Lambung Mangkurat University and graduated in 2011. After completing his undergraduate program, he worked as a software developer to gather experience for several years. He developed a lot of software, especially for local governments. In 2017, He completed his master's degree in Informatics from STMIK Amikom University. Currently, he is a lecturer in the Faculty of Mathematics and Natural Science at Lambung Mangkurat University. His research interests include software engineering, software defect prediction, and deep learning. He can be contacted at email: rudy.herteno@ulm.ac.id.

Friska Abadi finished his bachelor's degree in Computer Science from Lambung Mangkurat University in 2011. Subsequently, in 2016, he obtained his master's degree from the Department of Informatics at STMIK Amikom, Yogyakarta. Following that, he joined Lambung Mangkurat University as a lecturer in Computer Science. As a lecturer he teaches programming. Apart from that, he also carries out research and community service. Other activities as an application developer, whether using a web or mobile platform. Currently, he holds the position of head of the software engineering laboratory. His current area of research revolves around software engineering and also interested in machine learning.