


Article

Software Defect Prediction Using Dagging Meta-Learner-Based Classifiers

Akinbowale Nathaniel Babatunde ¹, Roseline Oluwaseun Ogundokun ^{2,3} , Latifat Bukola Adeoye ⁴ and Sanjay Misra ^{5,*}

¹ Department of Computer Science, Kwara State University, Ilorin 241103, Nigeria; akinbowale.babatunde@kwasu.edu.ng

² Department of Multimedia Engineering, Kaunas University of Technology, 44249 Kaunas, Lithuania; rosogu@ktu.lt or ogundokun.roseline@lmu.edu.ng

³ Department of Computer Science, Landmark University, Omu Aran 251103, Nigeria

⁴ Department of Computer Science, University of Ilorin, Ilorin 240003, Nigeria; adeoye.lb@unilorin.edu.ng

⁵ Department of Applied Data Science, Institute of Energy Technology, 1777 Halden, Norway

* Correspondence: sanjay.misra@ife.no

Abstract: To guarantee that software does not fail, software quality assurance (SQA) teams play a critical part in the software development procedure. As a result, prioritizing SQA activities is a crucial stage in SQA. Software defect prediction (SDP) is a procedure for recognizing high-risk software components and determining the influence of software measurements on the likelihood of software modules failure. There is a continuous need for sophisticated and better SDP models. Therefore, this study proposed the use of dagging-based and baseline classifiers to predict software defects. The efficacy of the dagging-based SDP model for forecasting software defects was examined in this study. The models employed were naïve Bayes (NB), decision tree (DT), and k-nearest neighbor (kNN), and these models were used on nine NASA datasets. Findings from the experimental results indicated the superiority of SDP models based on dagging meta-learner. Dagging-based models significantly outperformed experimented baseline classifiers built on accuracy, the area under the curve (AUC), F-measure, and precision-recall curve (PRC) values. Specifically, dagging-based NB, DT, and kNN models had +6.62%, +3.26%, and +4.14% increments in average accuracy value over baseline NB, DT, and kNN models. Therefore, it can be concluded that the dagging meta-learner can advance the recognition performances of SDP methods and should be considered for SDP processes.

Keywords: classification algorithm; defect prediction; software quality assurance; dagging meta-learner

MSC: 68Txx; 68T01



Citation: Babatunde, A.N.; Ogundokun, R.O.; Adeoye, L.B.; Misra, S. Software Defect Prediction Using Dagging Meta-Learner-Based Classifiers. *Mathematics* **2023**, *11*, 2714. <https://doi.org/10.3390/math11122714>

Academic Editors: Rafael Pastor, Bruno Domenech and Marc Juanpera

Received: 5 May 2023

Revised: 5 June 2023

Accepted: 13 June 2023

Published: 15 June 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

A structured method is employed in programming and software development in software engineering to enhance quality, time, and budget efficiency, including guaranteeing a structured software system [1–3]. The software procedure, similarly, recognized as the software method, is a series of organized activities that result in software conception. These tasks could comprise developing software from scratch or modifying a prevailing one [4]. The following actions must be included in any software development process. The first is the software specification, which outlines the program's core functionality and the limitations surrounding them. The second step is to design and create the software. Third, software verification and validation: the program must fulfill client requirements and correspond to its specifications. Finally, software evolution and maintenance involve changing the program to suit changing consumer and market desires [5]. They include sub-events, such as requirement authentication, architectural strategy, testing, and supportive events, such as configuration and modification administration, quality guarantee,

and project administration [6–8]. Other actions involve providing new technological tools, following best practices, and standardizing processes. A process, therefore, comprises the development description, which contains the following products: pre- and post-condition: the condition must be factual before and after the activity, the result of an action, and roles: the duties of the persons participating in the process [4].

A software flaw is any error or defectiveness in a software product or software development (SD), often known as a fault or bug [9–11]. Software defect prediction is one of the most practical tasks in the testing process, which recognizes the segments that are problem-predisposed to and need rigorous testing [12–14]. This allows experiment resources to be utilized more effectively while adhering to restrictions. Software defect prediction is beneficial during testing since it is not always possible to foresee the problem modules. Various difficulties, as well as the usage of faulty prediction models, obstruct smooth functioning [15–17].

It is essential to determine the software's defectiveness to organize testing operations since it is conceivable to concentrate further on the flaw-prone modules with the most mistakes. As a result, the testing process takes less time and effort, and the project's overall cost is reduced [18,19]. Software defect prediction models aim to forecast quality aspects, such as whether an element is susceptible to failure. Approaches for detecting flaw-prone software elements aid resource preparation and development, including cost elusion via efficient authentication [20,21]. These approaches may be employed to forecast the response variable: a module's class (e.g., flaw-prone or non-flaw-prone) or a quality factor (e.g., the number of flaws). To forecast software flaws, statistical tools, machine learning (ML) methods, and soft computing procedures are often utilized [22–25].

As a classification task, SDP may be considered as a supervised binary classification delinquent [26]. Software segments are labeled as faulty or non-defective and are represented by software metrics [27–29]. To train flaw analysts, historical data tables are created with one column containing a Boolean value for "flaws found" (i.e., reliant variable) and other columns describing software features concerning software measurement (i.e., independent variables) [30]. Grounded on a training set of data encompassing occurrence (or instances) whose class membership is recognized, binary classification in machine learning identifies which set of classes (sub-populations) a novel remark fits into [31]. Therefore, this research proposes the deployment of dagging meta-learner on classification algorithms for SDP processes. An experiment was designed to examine the efficacy of dagging-based SDP models for identifying software defects. Dagging-based and baseline classifiers, including NB, DT, and kNN, were utilized on nine NASA datasets. The following is the contribution of the study:

- i. Deployment of dagging meta-learners on classification algorithms.
- ii. Dagging-based and baseline classifiers, such as NB, DT, and kNN were applied to nine NASA datasets.
- iii. Use of an AUC, F-measure, and PRC value for system evaluation.

The remaining sections of this article are structured as follows: Section 2 offered the related literature on defect predictors that researchers have examined. Section 3 discussed the proposed system and the evaluation carried out to check the system's performance. The result obtained from the implementation and discussion on the results is presented in Section 4. Section 5 concludes the article and suggests upcoming works.

2. Related Works

Machine learning and deep learning have been used in different domains for optimization, scheduling, etc. [32–35]. Even though several defect prognosticators exist in the literature, few comprehensive benchmarking studies exist. Comparing the defect predictors' accuracy is crucial since the findings of one technique are seldom consistent across multiple datasets [36]. This is due to several factors. First, early defect prediction research relied on a limited number of datasets. Furthermore, since the performance metrics employed in each study differed, comparing them was challenging. Consequently, thorough benchmarking

studies are usually appreciated to determine which defect prediction approaches offer the most accurate results.

In their study, Xie et al. [37] use a multi-granularity neighbor residual network (MGNRN) to create an anomaly detection strategy for time-series data. They first establish a neighbor-based input matrix by considering multi-granularity neighborhood characteristics and build a neighbor input vector with a sliding time window for each data sample. Second, they use multi-granularity time windows to calculate the sample's linear and nonlinear neighbor characteristics. Finally, they anticipate the sample's anomalous probability by combining the linear neighborhood residual with the nonlinear residual. The precision and F1-metrics demonstrate the multi-granularity neighbor residual network's efficacy in improving the accuracy of anomalous detection, and the experiments support these claims.

Khurma et al. [38] proposed an island BMFO (IsBMFO) model. They presented an efficient binary form of MFO (BMFO). IsBMFO separates the population's solutions into islands, which are sub-populations. Each island is handled separately with a BMFO version. After a specific sum of iterations, a migration step was carried out to trade solutions across islands, increasing the algorithm's diversity power. The suggested strategy was evaluated using twenty-one publicly available software datasets. The trials indicated that employing IsBMFO as feature selection (FS) enhances the classification results. With an average G-mean of 78 percent, IsBMFO with SVM was deduced to be the best model for the SDP issue among the further analyzed techniques.

In a stringent CPDP environment, Malhotra, Khan, and Khera [39] developed a testing approach based on six distinct neural networks (NNs) on a dataset, including 20 softwares from the PROMISE repository. The optimum design was then compared with three suggested CPDP approaches that span a wide variety of circumstances. They discovered that the modest NN with dropout layers (NN-SD) was accomplished superlatively and statistically considered superior to the CPDP approaches when compared with other approaches throughout their investigation. The AUC for receiver operating characteristics (ROC) was employed as the performance measure. The Friedman chi-squared and Wilcoxon signed-rank tests were used to assess statistical implications.

To lower the impact of the class imbalance dataset, Jin [40] developed a unique distance-measure learning built on cost-sensitive learning (CSL), which is used with the large distribution machine (LDM) to replace the standard kernel function. Furthermore, the enhancement and enrichment of LDM based on CSL were investigated, with the improved LDM serving as the SDP model, dubbed CS-ILDM. The projected CS-ILDM was then employed with five publicly accessible datasets from NASA's Metrics Data Program repository (NASA MDPR), and its performance is similar to that of competing SDP approaches. The experimental findings show that the suggested CS-ILDM has high recognition performance, can lower the misprediction rate, and eliminate the influence of sample class imbalance.

Wang et al. [41] proposed an SDP system built on LASSO-SVM. The issue of most SDP models having low prediction accuracy was discussed in their work. A support vector machine technique and a least absolute value compression (LAVC), and a selection process are combined in an SDP model. First, the FS capability of the LAVC and selection technique was employed to minimize the dimension of the original dataset and non-SDP data were deleted. Then, utilizing the constraint optimization capability of the cross-validation technique, the optimum SVM value was determined. SVM's nonlinear computing capability completed the SDP. The recall rate was 78.04 percent, and the F-metric was 72.72 percent. The accuracy of simulation results was 93.25 percent and 66.67 percent. The findings revealed that the suggested defect prediction model outperforms the classic defect prediction model's prediction accuracy and speed.

Kumar and Shankar [42] created a Mamdani fuzzy logic-based software defect prediction model that predicted software faults using classic membership functions (Triangular, Trapezoidal, etc.) and a domain expert's unique membership function. They used a basic

Takagi–Sugeno model to enhance the Mamdani system and achieve better results. The plan assessed fuzzy logic models using standard regression models, such as multiple linear regression and random forest regression.

Akintola et al. [43] considered the impact of filter FS (FFS) on SDP classifiers. Ten NASA datasets (MW1, MC1, MC2, PC1, PC2, PC3, PC4, KC1, KC2, KC3), FFS algorithms including principal component analysis (PCA), filter subset evaluation (FSE), and correlation feature selection (CFF) subset evaluation including machine learning (ML) classification algorithms, such as NB, DT J48, MLP, and kNN were categorized using classifiers that had been carefully chosen based on their properties. According to their findings, feature selection methods can improve the performance of learning algorithms in SDP by eliminating immaterial or redundant features from the data ahead of the classification procedure. Nonetheless, the limitation of this study is that they only looked at filter-based feature selection, which is not the only type of feature selection.

Ranveer and Hiray [44] summarized malware detection (MD) methods founded on the stationary, active, and hybrid executable investigation. A comparative analysis of characteristics was offered, illuminating their impact on the system's performance. They discovered that using a suitable feature extraction strategy may result in high accuracy and an actual positive rate (TPR). Although op-code and portable executable (PE) capabilities improve the speed and accuracy of a malware detection system, false positives are still a problem. The suggested malware categorization algorithms should be able to cope with many daily new malware variants while maintaining system performance and accuracy in real time. However, the authors did not explore FS methods since the feature extraction technique would have changed the dataset's depiction.

Laradji et al. [45] explored multiple FS strategies for SDP. They found that picking a few high-quality features resulted in a substantially better AUC than using a more significant number of features. They similarly demonstrated the effectiveness of ensemble learning (EL) on datasets with unbalanced duplicated features. They suggested the utilization of a two-variant EL classifier. Experiments on six datasets revealed that greedy forward selection (GFS) significantly performed better than correlation-based forward (CBF) selection.

Additionally, we showed the efficacy of utilizing an average probability ensemble (APE) made up of seven properly designed learners, which outperformed traditional approaches, such as weighted SVMs and RF. Finally, the improved form of the suggested method, which coupled APE with GFS, achieved better AUC values for all datasets, which were near 1.0 in the case of the MC1, PC4, and PC2 datasets. However, for feature selection, the researchers only evaluated GFS and CBF, which is insufficient for a universal outcome.

He et al. [46] published empirical research on how a prognosticator founded on a reduced measured set was created and utilized for both with-in-project defect prediction (WPDP) and cross-project defect prediction (CPDP). The findings showed that the prognosticator created with a reduced measured set functioned well and that the suggested predictor and other benchmark predictors had no significant differences. The minimal metric subset was considered excellent based on the unique criteria for complexity, generalization, and accuracy since it may deliver good results in various circumstances and was independent of classifiers. In conclusion, their findings demonstrated that a more straightforward measure proposed for flaw forecast is realistic and valuable. A forecasting method built using a minimal subset of software measurement may deliver acceptable results. The investigators solely looked at feature selection filter approaches, with wrapper and hybrid FS being shown to be superior to the filter technique. The summary of the related works is shown in Table 1.

Table 1. Summary of related works.

Authors	Methods	Evaluation Measures	Gaps
Xie et al. [37]	Multi-granularity neighbor residual network (MGNRN)	F1-metrics and precision	The study did not consider semantic correlations of time stamps in the whole dataset to capture the global relevant features for the sample.
Khurma et al. [38]	Island BMFO (IsBMFO)	Average G-mean = 78%	IsBMFO as feature selection (FS) enhances the classification results.
Malhotra et al. [39]	6 NNs	AUC, Friedman chi-squared test, and the Wilcoxon signed-rank test	Statistical significance was evaluated.
Jin [40]	LDM based on CSL		The study can lower the misprediction rate and eliminate the influence of sample class imbalance.
Wang et al. [41]	LASSO-SVM	The recall rate was 78.04 percent, and the F-metric was 72.72 percent. The accuracy of simulation results was 93.25 percent and 66.67 percent.	The findings revealed that the suggested defect prediction model outperforms the classic defect prediction model's prediction accuracy and speed.
Kumar and Shankar [42]	Mamdani fuzzy logic	Not specified	They used a basic Takagi–Sugeno model to enhance the Mamdani system and achieve better results.
Akintola et al. [43]	Principal component analysis (PCA), filter subset evaluation (FSE), correlation feature selection (CFF) subset, NB, DT J48, MLP, and kNN	Not specified	According to their findings, feature selection methods can improve the performance of learning algorithms in SDP by eliminating immaterial or redundant features from the data ahead of the classification procedure.
Ranveer and Hiray [44]		TPR	They discovered that using a suitable feature extraction strategy may result in high accuracy and an actual positive rate (TPR).
Laradji, Alshayeb, and Ghouti [45]	Greedy forward selection (GFS), ensemble learning (EL), average probability ensemble (APE), correlation-based forward (CBF)	AUC	The improved form of the suggested approach, which coupled APE with GFS, achieved better AUC values for all datasets, which were near 1.0 in the case of the MC1, PC4, and PC2 datasets.
He et al. [46]	With-in project defect prediction (WPDP) and cross project defect prediction (CPDP)	Accuracy, complexity	The findings showed that the prognosticator created with a reduced measured set functioned well and that the suggested predictor and other benchmark predictors had no significant differences.

From the literature reviews discussed above, it is evident that several researchers have proposed and developed many approaches and techniques for SDP. Many have used techniques, such as ML, DM, etc. Many of these researches have low accuracy, precision, F-measure, and AUC ROC values. It was also discovered that many researchers did not even evaluate performance measures, such as AUC ROC values and F-measures. However, there is a continuous and imperative need to research and develop more accurate and sophisticated SDP models or methods, which led to the motivation behind this proposed study. In this study, we proposed the use of four performance measures to evaluate the system performance: Classification accuracy, precision, F-measure, and AUC ROC values.

3. Materials and Methods

This section discusses materials, like datasets, and methods, such as algorithms. This suggested system aims to examine and evaluate the impact of various wrapper feature selections (way of feature selection) on classifier performance for software fault detection. Decision tree (J48), naïve Bayes (NB), and k-nearest neighbor are the classifiers (kNN). All algorithms will be implemented by creating a route to the WEKA (Waikato Environment for Knowledge Analysis) API in the Eclipse IDE. Eclipse software was used to implement the method in this study. The experiment compares single classifiers versus dagging-based classifiers for software fault prediction using ANP. The datasets and experimental design are presented in this section. Nine public-domain software fault datasets were collected from NASA's MDP repository for this investigation. These nine public domain faults are PC1, PC3, PC4, PC5, CM1, KC1, KC3, MC2, and MW1. Each dataset was analyzed using 10-fold cross-validation, in which the dataset was divided into ten subsets, nine of which were used to train the classifier, and one subset was used to test the model generated by the classifier. A classifier's efficiency may be measured in a variety of ways. Accuracy, the area under the curve (AUC), and F-measure scores will all be assessed in this investigation [47]. The NASA MDP dataset used for the experimental analysis in this study was obtained from NASA MDP software defect datasets (figshare.com. Accessed on 17 June 2021).

3.1. Data Description

The National Aeronautics and Space Administration (NASA) Facility Metrics Data Program (MDP) repository contributed nine public-domain software defect datasets for this investigation. The following are short summaries of the MDP datasets, and the dataset attributes description is shown in Table 2.

1. PC1: This collection contains flight software from a decommissioned earth-orbiting satellite. It comprises 40 kilobytes of C code, 1107 modules, and 22 characteristics.
2. PC2: This dataset comes from flight software from a decommissioned earth-orbiting spacecraft. It comprises 40 kilobytes of C code, 1107 modules, and 22 characteristics.
3. PC3: This collection contains flight software from a presently active earth-orbiting satellite. It includes 1563 instances and 40 KLOC of C code.
4. PC4: This collection contains flight software from a presently active earth-orbiting satellite. It has 36 kilobytes of C code and 1458 modules.
5. CM1: This dataset comes from a research tool with around 20 kilo-source lines of code written in C code (KLOC). Overall, there are 498 occurrences and 22 characteristics.
6. MC2: This dataset has 161 occurrences and 40 characteristics.
7. MW1: This dataset is approximately a zero-gravity combustion research developed in C code with 8 KLOC and 403 modules.
8. KC1: This dataset comprises a C++-based stowage administration system for achieving and handing out data. Overall, there are 2109 instances and 22 characteristics.
9. KC3: This dataset is approximately the satellite metadata group, handing out, and dissemination. There are 458 instances and the it is inscribed in Java with 18 KLOC, 40 characteristics, and 18 KLOC.

Table 2. Dataset attributes description.

S/N	Attribute Name	Type
1	iv(g)	Numeric
2	Loc	Numeric
3	N	Numeric
4	ev(g)	Numeric
5	V	Numeric
6	E	Numeric
7	D	Numeric
8	B	Numeric
9	L	Numeric
10	I	Numeric
11	T	Numeric
12	ExecutableLoc	Numeric
13	IOCode	Numeric
14	IOBlank	Numeric
15	uniq_Opnd	Numeric
16	IOComment	Numeric
17	branchCount	Numeric
18	uniq_Op	Numeric
19	total_Opnd	Numeric
20	uniq_Opnd	Numeric
21	total_Op	Numeric
22	IOCodeAndComment	Numeric
23	Halstead level	Numeric
24	Call pairs	Numeric
25	Halstead vocabulary	Numeric
26	Halstead error	Numeric
27	Defects	categorical
28	Condition count	Numeric
29	Design density	Numeric
30	Edge count	Numeric
31	node count	Numeric
32	Maintenance severity	Numeric
33	Cyclomatic density	Numeric
34	Design complexity	Numeric
35	Modified condition count	Numeric
36	Decision density	Numeric
37	Formal parameters	Numeric

3.2. Proposed Models Implemented

Three classifiers were proposed in this study, and they include naïve Bayes (NB), decision tree (DT), and k-nearest neighbor (kNN). They are popular machine learning

algorithms used in various domains, including software defect prediction. These three models are discussed as follows:

3.2.1. Decision Tree (DT)

Decision trees are a categorization technique for organizing data [48,49]. The decision tree learns how the attribute vectors act in different situations. As described by Sandeep and Sharath [49], a decision tree is a graph in which each internal node represents a choice, and each child node represents the potential outcomes of that decision. The pathways from the tree's root to its leaves represent the problem's solutions. The tree-building and tree-pruning stages of the DT classification method are necessary. When generating a tree, data are partitioned recursively until each item has a single class label; when pruning, the created tree is reduced in size to avoid overfitting and boost its accuracy from the bottom up [49,50], see Algorithm 1.

Algorithm 1: Decision Tree (DT)

```

1.  if D contains only training examples of the same class  $c_j \in C$  then
2.      make T a leaf node labeled with class  $c_j$ 
3.  else if A = null then
4.      make T a leaf node labeled with  $c_j$ , which is the most frequent class in D
5.  else //D contains examples of a mixture of classes. We select a single attribute
6.      // to partition D into subsets in order that each subgroup is purer
7.       $p_o = \text{impurityEval-1}(D)$ ;
8.      for each attribute  $A_1 \in \{A_1, A_2, \dots, A_k\}$  perform
9.           $p_i = \text{impurityEval-1}(D)$ ;
10.         end
11.         Select  $A_g \in \{A_1, A_2, \dots, A_k\}$  that gives the biggest impurity reduction,
12.         if  $p_o - p_g < \text{threshold}$  then //  $A_g$  does not reduce impurity  $p_o$ 
13.             Make T a leaf node labeled with  $c_j$ , the most frequent class in D
14.         else
15.             Make T a decision node on  $A_g$ .
16.             Let the possible values of V be  $v_1, v_2, \dots, v_m$ .
17.             Partition D into m disjoint Subsets  $D_1, D_2, \dots, D_m$  based on m values of  $A_g$ .
18.             for each  $D_j$  in  $\{D_1, D_2, \dots, D_m\}$  perform
19.                 If  $D_j$  is not null then
20.                     Create a branch (edge) node  $T_j$  for  $v_j$  as a child node of T;
21.                     DecisionTree ( $D_j, A - \{A_g\}, T_j$ ) //  $A_g$  is removed
22.                 end
23.             end
24.         end

```

Decision tree Algorithm [51]

3.2.2. K-Nearest Neighbor (KNN)

Similarity-based instance classification is the goal of instance base learner (IBL) or k-nearest neighbor classification [49]. The function is merely approximated locally, and all computation is delayed until classification in this lazily-learned approach [49]. The majority of an object's neighbors determine its classification. Since K is always positive, the neighbors are picked from a pool of items whose labels are already known. To assign a class to a new data point, the k-nearest neighbors of that point in the training data are consulted [52,53]. The approach may be used to target functions with continuous or actual values [49], see Algorithm 2.

Algorithm 2: K-Nearest Neighbor

```

BEGIN
Input:
Build the training dataset  $D_i = \{ (X_1, C_1), \dots, (X_N, C_N) \}$ 
 $X = (X_1, \dots, X_N)$  new instance to be classified
For each labeled instance  $(X_i, C_i)$ , perform
If  $X$  has an unknown system call, then
 $X$  is abnormal;
else then
For each process,  $D_j$  in training data perform
calculate  $\text{Sim}(X, D_j)$ ;
if  $\text{Sim}(X, D_j) == 1.0$  then
 $X$  is normal and exit;
Order  $\text{Sim}(X, D_i)$  from Lowest to highest,  $(i = 1, \dots, N)$ ;
Find  $K$  biggest scores of  $\text{Sim}(X, D)$ ;
Select the  $K$  nearest instances to  $X$ :  $DKX$ ;
Assign to  $x$  the most frequent class in  $DKX$ ;
Calculate  $\text{Sim\_Avg}$  for  $k$ -nearest neighbors;
If  $\text{Sim\_Avg} > \text{threshold}$  then
 $X$  is normal;
else then
 $X$  is abnormal;
END
kNN Algorithm [54]

```

3.2.3. Naïve Bayes (NB)

The naïve Bayes algorithm is both a probabilistic classifier and a statistical classification technique; it determines a set of probabilities based on a dataset's frequency and combinations of values. Based on the Bayes theorem, this approach treats each attribute as independent of the value of the class variable [55,56]. It relies on the assumption of an underlying probabilistic model and provides a logical means of capturing uncertainty employing calculated probabilities. Moreover, it helps with both diagnosis and forecasting. Naïve Bayes is based on the Thomas Bayes (1702–1761) theorem and works best when the input space has many dimensions. In addition, naïve Bayes models employ the maximum likelihood for parameter estimation. Despite its oversimplified assumptions, naïve Bayes routinely outperforms more sophisticated machine learning algorithms in complex real-world settings.

3.2.4. Waikato Environment for Knowledge Analysis (WEKA)

To provide academics with simple access to cutting-edge machine learning methods, the Waikato Environment for Knowledge Analysis (WEKA) was developed. In 1992, when the project began, there was a wide selection of languages, systems, and data types supported by learning algorithms. Compiling learning schemes for comparison research across many datasets was an intimidating endeavor. In addition to a collection of learning algorithms, WEKA was designed to serve as a framework in which researchers may test and deploy novel algorithms without worrying about the underlying mechanisms required to manipulate data or evaluate the efficacy of proposed solutions [57,58].

The algorithms were implemented in Eclipse by building a path to WEKA (Waikato Environment for Knowledge Analysis) API library on Eclipse. WEKA is an open-source Java software that includes a collection of several machine learning algorithms for data analysis. The algorithms can be applied directly to a dataset or called from your Java code (the technique used in this project). Moreover, WEKA contains tools for performing data preprocessing, classification, association, regression, and visualization.

3.3. Motivations for Using the Proposed Models

The reasons for using these models for SDP are as follows:

Naïve Bayes, a probabilistic classifier, works on the feature independence assumption. It works effectively on massive datasets and requires little computer resources. When working with textual data, such as source code metrics or defect descriptions, NB excels due to its ability to cope with high-dimensional feature spaces. Due to the probabilistic nature of the algorithm, it can make accurate predictions quickly.

Decision trees are flexible models that may be easily understood and used for numerical and categorical information. They have the potential to capture intricate interrelationships between characteristics. DT is helpful for software defect prediction since they lighten the crucial aspects and how they affect the forecast. The resultant tree structure is simple to understand and relay to relevant parties, facilitating decision-making and troubleshooting.

Instances are sorted into groups using the k-nearest neighbor non-parametric approach based on their spatial closeness to other groups in the feature space. There is no implicit data distribution assumption. When software faults cluster in the same geographical area, kNN is an appropriate method for defect prediction. It can create accurate predictions based on closest neighbors by accurately capturing the similarity between occurrences. KNN is particularly flexible in that it can process data with both numerical and categorical characteristics.

A combination logic used the best features of all three models. The three models' predictions were integrated using dagging learning, which improved the total performance.

3.4. Proposed Architecture

The study's proposed architecture is shown in Figure 1. The methodology established is designed to experimentally evaluate and validate the efficacy of the offered approaches. The experimental framework is applied to nine NASA software defect datasets, and SDP models are created using K-fold ($k = 10$) cross-validation (CV). Due to its ability to construct phishing models with low bias and variance, the K-fold CV is preferred [59]. Furthermore, with the CV approach, each instance from a dataset is repeatedly utilized for training and testing. The CV approach is explained in detail in [60,61]. With faulty datasets based on a 10-fold CV, the suggested approaches and basic classifiers (NB, DT, and kNN) are trained and evaluated. The created SDP models' prediction performance is assessed and examined. The suggested approaches are implemented using WEKA machine learning tools and libraries [57].

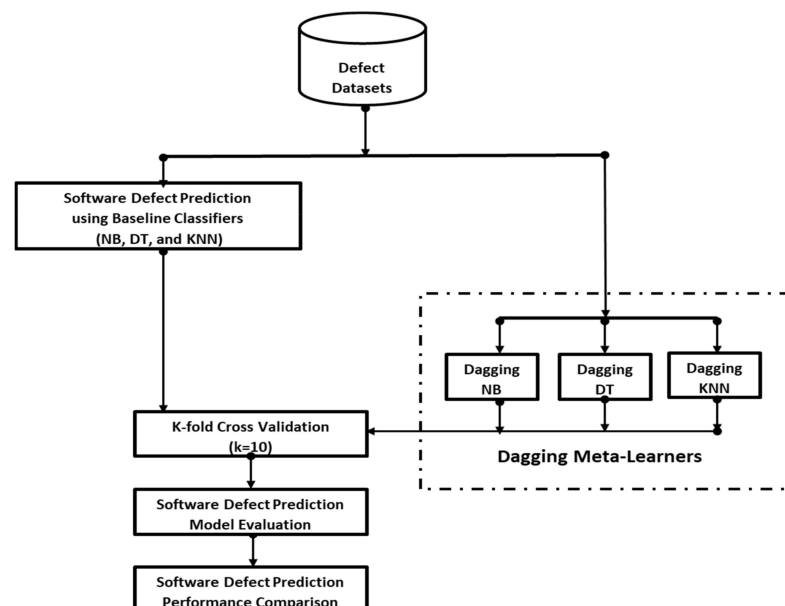


Figure 1. Experimental framework.

4. Results and Discussion

This study seeks to investigate the effect of wrapper feature selection methods on heterogeneous classifiers in SDP. This experiment was carried out by implementing the WEKA library using Eclipse IDE. This chapter presents the results generated after carrying out the research work. The tools (Eclipse and WEKA Library) used in carrying out this project work are open-source tools, and they can run on both Windows and Linux Operating Systems, a system with a minimum of 50 Gigabyte Hard Disk, and finally, a system with a minimum of 2 Gigabyte RAM (memory).

Tables 3–6 demonstrate the average prediction performance outcomes of experimented SDP models used over nine datasets, each divided into train and test datasets. Each table illustrates the results of the dagging-based classifiers and baseline classifiers.

Table 3 displays the accuracy values of a particular heterogeneous classifier and proposed dagging-based classifiers over nine SDP datasets. Table 3 shows some observations: Concerning accuracy, the dagging-based classifiers are better than baseline classifiers. In particular, Dagging_NB had an average accuracy value of 0.838%, which is superior to baseline NB (0.749%) by +11.1%. Moreover, Dagging_DT recorded an average accuracy value of 83.44%, which is +3.9% better than the baseline DT (0.803%). Dagging_kNN (0.832%) had a superior average accuracy value over baseline kNN (0.792%) by +5.1%. Based on the preceding experimental results, it can be deduced that dagging meta-learner-based classification can improve the prediction accuracy values of SDP models. Figure 2 presents a graphical representation of the average accuracy values of experimented SDP models in this study.

Table 3. Accuracy values of dagging-based classifiers and baseline classifiers.

AUC	NB	Dagging_NB	DT	Dagging_DT	kNN	Dagging_kNN
CM1	0.645	0.708	0.570	0.596	0.521	0.611
KC1	0.681	0.669	0.604	0.681	0.633	0.679
KC3	0.662	0.702	0.653	0.645	0.539	0.623
MC2	0.707	0.684	0.589	0.708	0.653	0.706
MW1	0.778	0.753	0.503	0.771	0.607	0.722
PC1	0.791	0.786	0.598	0.767	0.679	0.785
PC3	0.749	0.785	0.591	0.782	0.603	0.753
PC4	0.814	0.816	0.789	0.897	0.7	0.844
PC5	0.719	0.699	0.673	0.765	0.667	0.764
Average	0.727	0.734	0.619	0.735	0.622	0.721

Table 4. AUC values of dagging-based classifiers and baseline classifiers.

AUC	NB	Dagging_NB	DT	Dagging_DT	kNN	Dagging_kNN
CM1	0.645	0.708	0.570	0.596	0.521	0.611
KC1	0.681	0.669	0.604	0.681	0.633	0.679
KC3	0.662	0.702	0.653	0.645	0.539	0.623
MC2	0.707	0.684	0.589	0.708	0.653	0.706
MW1	0.778	0.753	0.503	0.771	0.607	0.722
PC1	0.791	0.786	0.598	0.767	0.679	0.785
PC3	0.749	0.785	0.591	0.782	0.603	0.753
PC4	0.814	0.816	0.789	0.897	0.7	0.844
PC5	0.719	0.699	0.673	0.765	0.667	0.764
Average	0.727	0.734	0.619	0.735	0.622	0.721

Table 5. F-measure values of dagging-based classifiers and baseline classifiers.

F-Measure	NB	Dagging_NB	DT	Dagging_DT	KNN	Dagging_KNN
CM1	0.816	0.806	0.802	0.815	0.782	0.815
KC1	0.718	0.723	0.717	0.714	0.727	0.715
KC3	0.785	0.755	0.783	0.747	0.708	0.763
MC2	0.672	0.702	0.608	0.69	0.704	0.632
MW1	0.842	0.947	0.896	0.869	0.845	0.890
PC1	0.894	0.892	0.901	0.888	0.906	0.887
PC3	0.485	0.777	0.839	0.839	0.843	0.836
PC4	0.841	0.837	0.869	0.856	0.857	0.840
PC5	0.704	0.717	0.737	0.734	0.741	0.738
Average	0.751	0.795	0.795	0.795	0.790	0.791

Table 6. Precision-recall curve values of dagging-based classifiers and baseline classifiers.

PRC	NB	Dagging_NB	DT	Dagging_DT	KNN	Dagging_KNN
CM1	0.825	0.850	0.790	0.808	0.781	0.818
KC1	0.741	0.733	0.670	0.743	0.691	0.730
KC3	0.777	0.802	0.763	0.767	0.713	0.757
MC2	0.718	0.690	0.601	0.712	0.643	0.728
MW1	0.893	0.891	0.824	0.900	0.843	0.887
PC1	0.918	0.921	0.861	0.909	0.888	0.911
PC3	0.861	0.885	0.796	0.882	0.813	0.863
PC4	0.882	0.896	0.861	0.920	0.828	0.894
PC5	0.761	0.756	0.698	0.791	0.700	0.797
Average	0.820	0.825	0.763	0.826	0.767	0.821

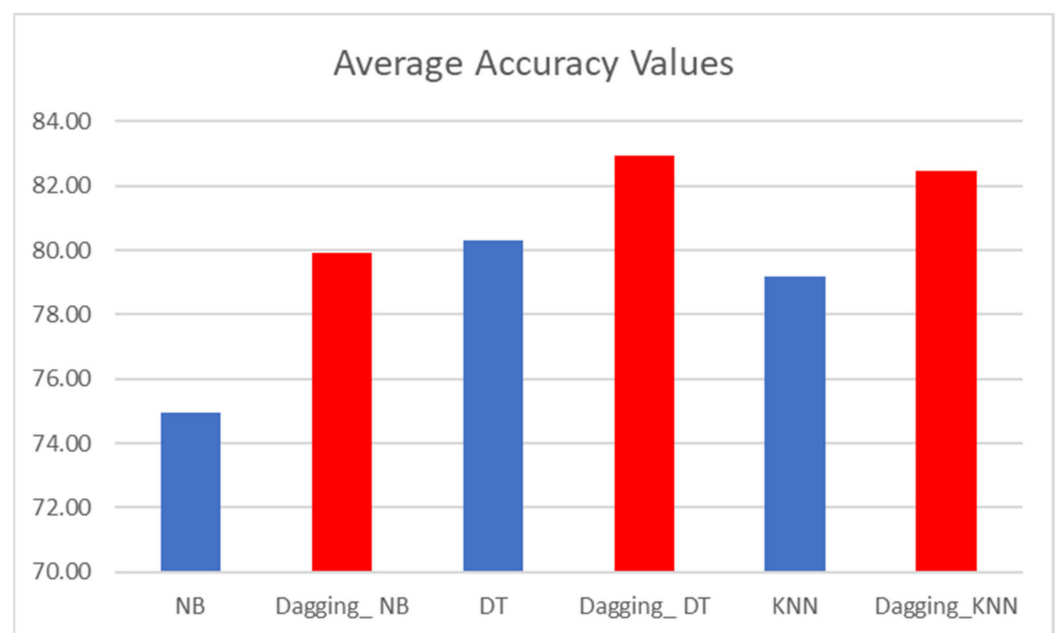


Figure 2. A graphical representation of experimental results based on average accuracy values.

Concerning AUC values, Table 4 presents the AUC values of dagging-based classifiers and baseline classifiers. Similar to the accuracy values, dagging-based SDP models have superior AUC values than models based on baseline (NB, DT, kNN) classifiers. Specifically, Dagging_NB had an average AUC value of 0.785, which is superior to baseline NB (0.727) by +7.98%. Furthermore, Dagging_DT recorded an average AUC value of 0.78, which is +26% better than the baseline DT (0.619). Dagging_KNN (0.777) had a superior average accuracy value over baseline kNN (0.622) by +24.9%. Correspondingly, it can be observed that dagging-based classification further improved the AUC values of SDP models. In addition, SDP models based on dagging had AUC values close to 1, namely, SDP models based on dagging have a high ability to predict defects and are not subject to change. This observation further strengthens the findings from Table 3, namely, SDP models based on dagging are a good fit. Figure 3 presents a graphical representation of the average AUC values of dagging-based classifiers and baseline SDP models.

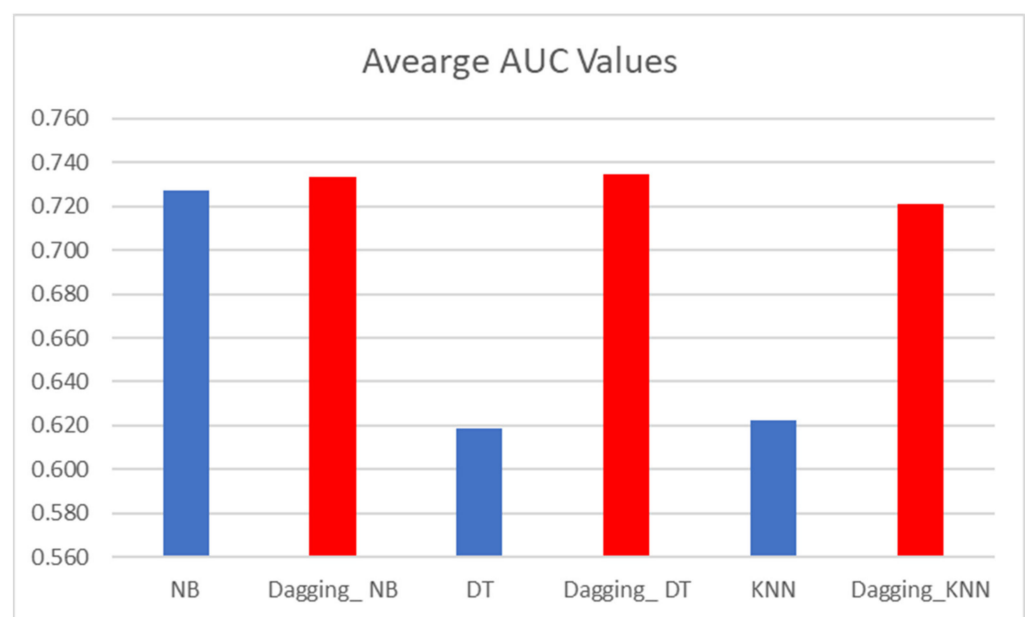


Figure 3. A graphical representation of experimental results based on average AUC values.

In terms of F-measure values, Table 5 shows the F-measure values of dagging-based classifiers and baseline classifiers. Moreover, similar to the initial results in Tables 3 and 4, dagging-based SDP models have better F-measure values than models based on baseline (NB, DT, kNN) classifiers. Specifically, Dagging_NB had an average F-measure value of 0.805, which is superior to baseline NB (0.751) by +7.19%. Moreover, Dagging_DT recorded an average AUC value of 0.808, which is +1.64% better than baseline DT (0.795). Dagging_KNN (0.790) had a superior average accuracy value over baseline kNN (0.807) by +2.15%. Correspondingly, it can be observed that dagging meta-learner classifier further enhanced the F-measure values of SDP models. This observation further strengthens and agrees with the findings from Tables 3 and 4. Similarly, Figure 4 presents a graphical representation of the average AUC values of dagging meta-learner and baseline SDP models.

Finally, Table 6 presents the precision-recall curve (PRC) values of the baseline and projected dagging-based classifiers. The PRC value is built on the precision and recall values of the developed SDP models. It recaps the trade-off between the TPR and the positive prognostic rate for various probability thresholds in a predictive model. It can also be deduced from Table 6 that SDP models founded on dagging meta-learner classifiers are better than baseline classifiers, in particular. Dagging_NB had an average accuracy value of 0.853, which is superior to baseline NB (0.820) by +4.02%. Moreover, Dagging_DT recorded an average accuracy value of 0.847, which was +11.1% better than baseline DT (0.763). Dagging_KNN (0.847) had a greater average F-measure rate over baseline kNN

(0.767) by +10.43%. Based on these investigational outcomes, it can be discovered that SDP models founded on dagging meta-learner classifiers are superior to SDP models based on baseline classifiers. A graphical representation of the PRC values is presented in Figure 5.

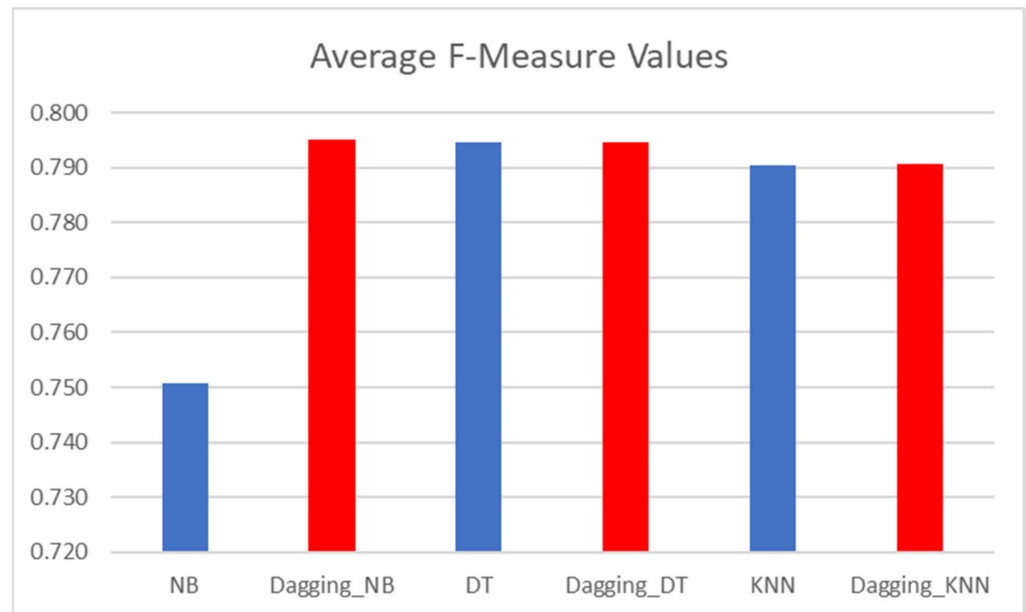


Figure 4. A graphical representation of experimental results based on average F-measure values.

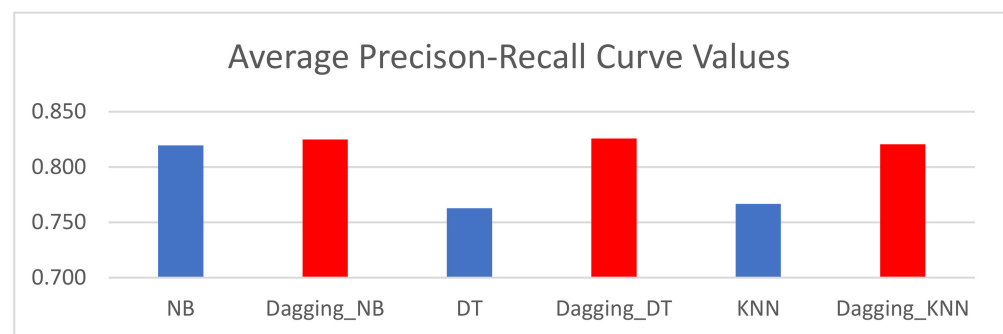


Figure 5. A graphical representation of experimental results based on average PRC values.

Compared to baseline classifiers, the high and better prediction performance of dagging meta-learner-based SDP models on the analyzed datasets indicates their corresponding small risk of misprediction of software flaws. Furthermore, the suggested approaches are extra resilient and robust to prejudices in the examined datasets (class imbalance and high dimensionality) than the baseline classifiers, as seen by the high AUC values of dagging meta-learner-based SDP models (NB, DT, and kNN). In particular, the suggested approaches outperform baseline classifiers in predicting software defects due to class imbalance and high-dimensionality issues that might be present in software defect datasets.

5. Threats to Validity

A number of the constraints encountered during the present investigation, such as those seen in earlier studies, are as follows: Validity is threatened by two categories of risks. The risk to external validity and the chance to internal validity. Threats to external validity are seen to be more significant than risks to internal validity. The degree to which inferences may be formed between independent and dependent variables poses a threat to internal validity [40]. It is possible that the data are not cumulative, and there is a gap between the numbers that must be handled. External validity risks are connected to the generalizability

of the anticipated models. The findings presented in this research were achieved using the open-source program WEKA tool, and thus may not be relevant to other systems. In particular, the operating environment determines the efficiency of dependability prediction models. However, the dataset is not significantly large. These risks may be reduced by performing more replicated experiments across several platforms. Ultimately, despite all of these limitations and restrictions, the results of our study provide direction for further research into the influence of prior failure datasets on software reliability prediction using machine learning methods.

6. Conclusions and Future Work

Software businesses must concentrate their limited SQA resources on software segments (such as source code files) that are probably problematic. Statistical or machine learning classification approaches are used to train defect forecasting methods to recognize fault-prone software segments. ML algorithms have varying levels of efficiency, which might vary depending on the performance measurements used and the conditions.

This study aimed to observe how successful dagging meta-learner-based SDP models predict software defect problems. Dagging-based classifiers and baseline classifiers, such as NB, DT, and kNN were used on nine NASA datasets. The experimental findings revealed that SDP models were built on dagging meta-learner beat-tested baseline classifiers regarding accuracy, AUC, F-measure, and PRC values. Dagging_NB had an average accuracy value of 83.75%, which is superior to baseline NB (74.93%) by +11.06%. Moreover, Dagging_DT recorded an average accuracy value of 83.44%, which is +3.91% better than the baseline DT (80.30%). Dagging_KNN (83.24%) had a superior average accuracy value over baseline kNN (79.17%) by +5.14%.

Similarly, Dagging_NB had an average AUC value of 0.785, which is superior to baseline NB (0.727) by +7.98%. Moreover, Dagging_DT recorded an average AUC value of 0.78, which is +26% better than baseline DT (0.619). Dagging_KNN (0.777) had a superior average accuracy value over baseline kNN (0.622) by +24.9%. Correspondingly, Dagging_NB had an average F-measure value of 0.805, which is superior to baseline NB (0.751) by +7.19%. Furthermore, Dagging_DT recorded an average AUC value of 0.808, which is +1.64% better than baseline DT (0.795). Dagging_KNN (0.790) had a superior average accuracy value over baseline kNN (0.807) by +2.15%.

Finally, it was observed that Dagging_NB had an average accuracy value of 0.853, which is superior to baseline NB (0.820) by +4.02%. Moreover, Dagging_DT recorded an average accuracy value of 0.847, which was +11.1% better than baseline DT (0.763). Dagging_KNN (0.847) had a greater average F-measure rate over baseline kNN (0.767) by +10.43%. Therefore, it can be concluded that since dagging meta-learner outperformed baseline classifiers in terms of accuracy, AUC, F-measure, and PRC values, it may be used to improve SDP model prediction performance and should be considered for SDP procedures.

In the future, more effective SDP approaches or processes can be developed by implementing deep learning algorithms, optimization algorithms, or even dimensionality-reduction algorithms. Several datasets can be gathered and validated by employing numerous other ML techniques for defect forecasting. More research is planned to integrate the abovementioned models with different machine learning approaches to produce prediction models that can forecast software dependability more correctly and with fewer accuracy errors.

Furthermore, we proposed the examination of recent emerging domains, such as tabu search with simulated annealing for solving a location-protection-disruption in hub network, dark-side avoidance of mobile applications with data biases elimination in the socio-cyber world, etc., in which recent methods, such as tabu search and anomaly detection will be researched in future research. Some other ML models, such as XGBoost, AdaBoost, and LightGBM, are proposed to be implemented in the future.

Author Contributions: Conceptualization, A.N.B., L.B.A. and S.M.; methodology, A.N.B., L.B.A. and S.M.; software, L.B.A.; validation, R.O.O. and S.M.; formal analysis, R.O.O.; investigation, S.M.; resources, L.B.A.; data curation, A.N.B. and L.B.A.; writing—original draft preparation, L.B.A.; writing—review and editing, R.O.O. and S.M.; visualization, R.O.O. and L.B.A.; supervision, S.M., A.N.B. and R.O.O.; project administration, A.N.B. and R.O.O.; funding acquisition, S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The dataset used for this study is available online at NASA MDP Software Defect Datasets (figshare.com). Accessed on 17 June 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lu, H.; Zhu, Y.; Yin, M.; Yin, G.; Xie, L. Multimodal Fusion Convolutional Neural Network With Cross-Attention Mechanism for Internal Defect Detection of Magnetic Tile. *IEEE Access* **2022**, *10*, 60876–60886. [[CrossRef](#)]
2. Song, Y.; Xin, R.; Chen, P.; Zhang, R.; Chen, J.; Zhao, Z. Identifying performance anomalies in fluctuating cloud environments: A robust correlative-GNN-based explainable approach. *Future Gener. Comput. Syst.* **2023**, *145*, 77–86. [[CrossRef](#)]
3. Chen, H.; Xiong, Y.; Li, S.; Song, Z.; Hu, Z.; Liu, F. Multi-Sensor Data Driven with PARAFAC-IPSO-PNN for Identification of Mechanical Nonstationary Multi-Fault Mode. *Machines* **2022**, *10*, 155. [[CrossRef](#)]
4. Braude, E.J.; Bernstein, M.E. *Software Engineering: Modern Approaches*; Waveland Press: Long Grove, IL, USA, 2016.
5. Abrahamsson, P.; Salo, O.; Ronkainen, J.; Warsta, J. Agile software development methods: Review and analysis. *arXiv* **2017**, arXiv:1709.08439.
6. Liu, X.; Li, Z.; Fu, X.; Yin, Z.; Liu, M.; Yin, L.; Zheng, W. Monitoring House Vacancy Dynamics in The Pearl River Delta Region: A Method Based on NPP-VIIRS Night-Time Light Remote Sensing Images. *Land* **2023**, *12*, 831. [[CrossRef](#)]
7. Lu, S.; Ding, Y.; Liu, M.; Yin, Z.; Yin, L.; Zheng, W. Multiscale Feature Extraction Fusion of Image Text in VQA. *Int. J. Comput. Intell. Syst.* **2023**, *16*, 54. [[CrossRef](#)]
8. Liu, X.; He, J.; Liu, M.; Yin, Z.; Yin, L.; Zheng, W. A Scenario-Generic Neural Machine Translation Data Augmentation Method. *Electronics* **2023**, *12*, 2320. [[CrossRef](#)]
9. Kumar, L.; Dastidar, T.G.; Murthy Neti, L.B.; Satapathy, S.M.; Misra, S.; Kocher, V.; Padmanabhuni, S. Deep-Learning Approach with DeepXplore for Software Defect Severity Level Prediction. In *International Conference on Computational Science and Its Applications*; Springer: Cham, Switzerland, 2021; pp. 398–410.
10. Choeikiwong, T.; Vateekul, P. Software defect prediction in imbalanced data sets using unbiased support vector machine. In *Information Science and Applications*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 923–931.
11. Kumar, L.; Misra, S.; Rath, S.K. An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. *Comput. Stand. Interfaces* **2017**, *53*, 1–32. [[CrossRef](#)]
12. Anand, A.; Agarwal, M.; Tamura, Y.; Yamada, S. Economic impact of software patching and optimal release scheduling. *Qual. Reliab. Eng. Int.* **2017**, *33*, 149–157. [[CrossRef](#)]
13. Lv, Z.; Kumar, N. Software defined solutions for sensors in 6G/IoE. *Comput. Commun.* **2020**, *153*, 42–47. [[CrossRef](#)]
14. Guo, F.; Zhou, W.; Lu, Q.; Zhang, C. Path extension similarity link prediction method based on matrix algebra in directed networks. *Comput. Commun.* **2022**, *187*, 83–92. [[CrossRef](#)]
15. Ullah, N. A method for predicting open-source software residual defects. *Softw. Qual. J.* **2015**, *23*, 55–76. [[CrossRef](#)]
16. Chen, H.; Liu, M.; Chen, Y.; Li, S.; Miao, Y. Nonlinear Lamb Wave for Structural Incipient Defect Detection with Sequential Probabilistic Ratio Test. *Secur. Commun. Netw.* **2022**, *2022*, 9851533. [[CrossRef](#)]
17. Huang, N.; Chen, Q.; Cai, G.; Xu, D.; Zhang, L.; Zhao, W. Fault Diagnosis of Bearing in Wind Turbine Gearbox Under Actual Operating Conditions Driven by Limited Data With Noise Labels. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 1–10. [[CrossRef](#)]
18. Jimoh, R.; Balogun, A.; Bajeh, A.; Ajayi, S. A PROMETHEE based evaluation of software defect predictors. *J. Comput. Sci. Its Appl.* **2018**, *25*, 106–119.
19. Mao, Y.; Zhu, Y.; Tang, Z.; Chen, Z. A Novel Airspace Planning Algorithm for Cooperative Target Localization. *Electronics* **2022**, *11*, 2950. [[CrossRef](#)]
20. Ren, Y.; Jiang, H.; Ji, N.; Yu, H. TBSM: A traffic burst-sensitive model for short-term prediction under special events. *Knowl.-Based Syst.* **2022**, *240*, 108120. [[CrossRef](#)]
21. Zhang, J.; Liu, Y.; Li, Z.; Lu, Y. Forecast-Assisted Service Function Chain Dynamic Deployment for SDN/NFV-Enabled Cloud Management Systems. *IEEE Syst. J.* **2023**, 1–12. [[CrossRef](#)]
22. Agarwal, S.; Tomar, D. Prediction of software defects using twin support vector machine. In Proceedings of the 2014 International Conference on Information Systems and Computer Networks (ISCON), Mathura, India, 1–2 March 2014.
23. Malhotra, R.; Sharma, A. Analyzing Machine Learning Techniques for Fault Prediction Using Web Applications. *J. Inf. Process. Syst.* **2018**, *14*, 751–770.

24. Garg, H. A Brief Analysis of Soft Computing Techniques in Software Fault Prediction. In Proceedings of the 2021 5th International Conference on Information Systems and Computer Networks (ISCON), Mathura, India, 22–23 October 2021; IEEE: New York, NY, USA, 2021; pp. 1–7.
25. Cheng, B.; Zhu, D.; Zhao, S.; Chen, J. Situation-Aware IoT Service Coordination Using the Event-Driven SOA Paradigm. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 349–361. [[CrossRef](#)]
26. Okutan, A.; Yıldız, O.T. Software defect prediction using Bayesian networks. *Empir. Softw. Eng.* **2014**, *19*, 154–181. [[CrossRef](#)]
27. Xiong, S.; Li, B.; Zhu, S. DCGNN: A single-stage 3D object detection network based on density clustering and graph neural network. *Complex Intell. Syst.* **2022**, *9*, 3399–3408. [[CrossRef](#)]
28. Wang, B.; Zhang, Y.; Zhang, W. A Composite Adaptive Fault-Tolerant Attitude Control for a Quadrotor UAV with Multiple Uncertainties. *J. Syst. Sci. Complex.* **2020**, *35*, 81–104. [[CrossRef](#)]
29. Tan, X.; Lin, J.; Xu, K.; Chen, P.; Ma, L.; Lau, R.W. Mirror Detection With the Visual Chirality Cue. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *45*, 3492–3504. [[CrossRef](#)] [[PubMed](#)]
30. Zemmal, N.; Azizi, N.; Sellami, M.; Zenakhra, D.; Cheriguene, S.; Dey, N.; Ashour, A.S. Robust feature selection algorithm based on transductive SVM wrapper and genetic algorithm: Application on computer-aided glaucoma classification. *Int. J. Intell. Syst. Technol. Appl.* **2018**, *17*, 310–346. [[CrossRef](#)]
31. Chang, R.; Shen, X.; Wang, B.; Xu, Q. A novel method for software defect prediction in the context of big data. In Proceedings of the 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), Beijing, China, 10–12 March 2017; IEEE: New York, NY, USA, 2017; pp. 100–104.
32. Yan, A.; Li, Z.; Cui, J.; Huang, Z.; Ni, T.; Girard, P.; Wen, X. LDAVPM: A Latch Design and Algorithm-based Verification Protected against Multiple-Node-Upsets in Harsh Radiation Environments. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2022**, *2022*, 2069–2073. [[CrossRef](#)]
33. Lu, C.; Zheng, J.; Yin, L.; Wang, R. An improved iterated greedy algorithm for the distributed hybrid flowshop scheduling problem. *Eng. Optim.* **2023**, *2023*, 1–13. [[CrossRef](#)]
34. Li, B.; Tan, Y.; Wu, A.; Duan, G. A distributionally robust optimization based method for stochastic model predictive control. *IEEE Trans. Autom. Control.* **2021**, *67*, 5762–5776. [[CrossRef](#)]
35. Lyu, W.; Wang, Z. Global classical solutions for a class of reaction-diffusion system with density-suppressed motility. *Electron. Res. Arch.* **2022**, *30*, 995–1015. [[CrossRef](#)]
36. Ghotra, B.; McIntosh, S.; Hassan, A.E. Revisiting the impact of classification techniques on the performance of defect prediction models. In Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy, 16–24 May 2015; IEEE: New York, NY, USA, 2015; Volume 1, pp. 789–800.
37. Xie, H.; Hao, C.; Li, J.; Li, M.; Luo, P.; Zhu, J. Anomaly Detection For Time Series Data Based on Multi-granularity Neighbor Residual Network. *Int. J. Cogn. Comput. Eng.* **2022**, *3*, 180–187.
38. Khurma, R.A.; Alsawalqah, H.; Aljarah, I.; Elaziz, M.A.; Damaševičius, R. An Enhanced Evolutionary Software Defect Prediction Method Using Island Moth Flame Optimization. *Mathematics* **2021**, *9*, 1722. [[CrossRef](#)]
39. Malhotra, R.; Aggarwal, D.; Garg, P. Application of Random Vector Functional Link Network for Software Defect Prediction. In *International Conference on Emerging Trends and Technologies on Intelligent Systems*; Springer: Singapore, 2021; pp. 127–143.
40. Jin, C. Software defect prediction model based on distance metric learning. *Soft Comput.* **2021**, *25*, 447–461. [[CrossRef](#)]
41. Wang, K.; Liu, L.; Yuan, C.; Wang, Z. Software defect prediction model based on LASSO-SVM. *Neural Comput. Appl.* **2021**, *33*, 8249–8259. [[CrossRef](#)]
42. Kumar, K.S.M. Software Defect Prediction with Fuzzy Logic. Ph.D. Thesis, Auburn University, Auburn, Alabama, 2020.
43. Akintola, A.G.; Balogun, A.O.; Lafenwa-Balogun, F.B.; Mojeed, H.A. Comparative Analysis of Selected Heterogeneous Classifiers for Software Defects Prediction Using Filter-Based Feature Selection Methods. *FUOYE J. Eng. Technol.* **2018**, *3*, 134–137. [[CrossRef](#)]
44. Ranveer, S.; Hiray, S. Comparative analysis of feature extraction methods of malware detection. *Int. J. Comput. Appl.* **2015**, *120*, 1–7. [[CrossRef](#)]
45. Laradji, I.H.; Alshayeb, M.; Ghouti, L. Software defect prediction using ensemble learning on selected features. *Inf. Softw. Technol.* **2015**, *58*, 388–402. [[CrossRef](#)]
46. He, P.; Li, B.; Liu, X.; Chen, J.; Ma, Y. An empirical study on software defect prediction with a simplified metric set. *Inf. Softw. Technol.* **2015**, *59*, 170–190. [[CrossRef](#)]
47. Ogundokun, R.O.; Awotunde, J.B.; Sadiku, P.; Adeniyi, E.A.; Abiodun, M.; Dauda, O.I. An Enhanced Intrusion Detection System using Particle Swarm Optimization Feature Extraction Technique. *Procedia Comput. Sci.* **2021**, *193*, 504–512. [[CrossRef](#)]
48. Kaur, G.; Chhabra, A. Improved J48 classification algorithm for the prediction of diabetes. *Int. J. Comput. Appl.* **2014**, *98*, 13–17. [[CrossRef](#)]
49. Patra, S.K.; Prasad, C.S.C.S. A Survey of Different Classification Techniques and Their Comparison Using Mc Nemar’s Test. Ph.D. Thesis, National Institute of Technology, Rourkela, India, 2013.
50. Ogundokun, R.O.; Misra, S.; Ogundokun, O.E.; Oluranti, J.; Maskeliunas, R. Machine learning classification based techniques for fraud discovery in credit card datasets. In *Applied Informatics, Proceedings of the Fourth International Conference, ICAI 2021, Buenos Aires, Argentina, 28–30 October 2021*; Springer: Cham, Switzerland, 2021; pp. 26–38.

51. Croce, P.R.; Júnior, R.G.D.S.V.; da Hora, H.R.M.; de Assis Rangel, J.J. South America Energy Matrix: A Decision Tree Approach. Available online: https://www.researchgate.net/profile/Paulo-Rossi-3/publication/346577865_SOUTH_AMERICA_ENERGY_MATRIX_A_DECISION_TREE_APPROACH/links/5fc84835a6fdcc697bd79c07/SOUTH-AMERICA-ENERGY-MATRIX-A-DECISION-TREE-APPROACH.pdf (accessed on 17 June 2021).
52. Ogundokun, R.O.; Arowolo, M.O.; Misra, S.; Damasevicius, R. An Efficient Blockchain-Based IoT System Using Improved KNN Machine Learning Classifier. In *Blockchain Based Internet of Things*; Springer: Singapore, 2022; pp. 171–180.
53. Guo, G.; Wang, H.; Bell, D.; Bi, Y.; Greer, K. KNN model-based approach in classification. In *On the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, Proceedings of the OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, 3–7 November 2003*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 986–996.
54. Liao, Y.; Vemuri, V.R. Use of k-nearest neighbor classifier for intrusion detection. *Comput. Secur.* **2002**, *21*, 439–448. [[CrossRef](#)]
55. Gbadamosi, B.; Ogundokun, R.O.; Adeniyi, E.A.; Misra, S.; Stephens, N.F. Medical Data Analysis for IoT-Based Datasets in the Cloud Using Naïve Bayes Classifier for Prediction of Heart Disease. In *New Frontiers in Cloud Computing and Internet of Things*; Springer: Cham, Switzerland, 2022; pp. 365–386.
56. Webb, G.I.; Keogh, E.; Miikkulainen, R. Naïve Bayes. *Encycl. Mach. Learn.* **2010**, *15*, 713–714.
57. Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I.H. The WEKA data mining software: An update. *ACM SIGKDD Explor. Newsl.* **2009**, *11*, 10–18. [[CrossRef](#)]
58. Oladele, T.O.; Ogundokun, R.O.; Kayode, A.A.; Adegun, A.A.; Adebisi, M.O. Application of data mining algorithms for feature selection and prediction of diabetic retinopathy. In *Computational Science and Its Applications—ICCSA 2019, Proceedings of the 19th International Conference, Saint Petersburg, Russia, 1–4 July 2019*; Proceedings, Part V 19; Springer: Cham, Switzerland, 2019; pp. 716–730.
59. Balogun, A.O.; Basri, S.; Abdulkadir, S.J.; Hashim, A.S. Performance analysis of feature selection methods in software defect prediction: A search method approach. *Appl. Sci.* **2019**, *9*, 2764. [[CrossRef](#)]
60. Arlot, S.; Lerasle, M. Choice of V for V-fold cross-validation in least-squares density estimation. *J. Mach. Learn. Res.* **2016**, *17*, 7256–7305.
61. Yadav, S.; Shukla, S. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In *Proceedings of the 2016 IEEE 6th International Conference on Advanced Computing (IACC), Bhimavaram, India, 27–28 February 2016*; IEEE: New York, NY, USA, 2016; pp. 78–83.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.