engineering proceedings

MDPI

Proceeding Paper

Developing a Machine Learning-Based Software Fault Prediction Model Using the Improved Whale Optimization Algorithm †

Hauwa Abubakar [1,*], Kabir Umar [2], Rukayya Auwal [1], Kabir Muhammad [1] and Lawan Yusuf [3]

[1] Department of Computer Science, Faculty of Computing, Bayero University, Kano 700006, Nigeria; rukayyaauwal@gmail.com (R.A.); kabirmuhammad3@gmail.com (K.M.)
[2] Department of Software Engineering, Faculty of Computing, Bayero University, Kano 700006, Nigeria; ukabir.se@buk.edu.ng
[3] Department of Information and Communication Technology, Directorate of Examination and Assessments, National Open University of Nigeria, Abuja 900001, Nigeria; lyusuf@noun.edu.ng
* Correspondence: hauwaabubakarmusa93@gmail.com
† Presented at the 4th International Electronic Conference on Applied Sciences, 27 October–10 November 2023; Available online: https://asec2023.sciforum.net/.

Abstract: Software fault prediction (SFP) is vital for ensuring software system reliability by detecting and mitigating faults. Machine learning has proven effective in addressing SFP challenges. However, extensive fault data from historical repositories often lead to dimensionality issues due to numerous metrics. Feature selection (FS) helps mitigate this problem by identifying key features. This research enhances the Whale Optimization Algorithm (WOA) by combining truncation selection with a single-point crossover method to enhance exploration and avoid local optima. Evaluating the enhancement on 14 SFP datasets from the PROMISE repository reveals its superiority over the original WOA and other variants, demonstrating its potential for improved SFP.

Keywords: software fault prediction; whale optimization; feature selection; machine learning; truncation selection

Citation: Abubakar, H.; Umar, K.; Auwal, R.; Muhammad, K.; Yusuf, L. Developing a Machine Learning-Based Software Fault Prediction Model Using the Improved Whale Optimization Algorithm. *Eng. Proc.* **2023**, *56*, 334. https://doi.org/10.3390/ASEC2023-16307

Academic Editor: Nunzio Cennamo

Published: 21 November 2023

1. Introduction

SFP greatly aids in producing high-quality software at a low cost by identifying fault-prone software modules [1]. Machine learning algorithms like decision trees, Bayesian learners, neural networks, support vector machines, and rule-based learning have shown promise, as have soft computing approaches like fuzzy computing, neural networks, evolutionary computing, and swarm intelligence [2].

Feature selection is frequently used to improve the SFP performance of machine learning (ML) algorithms, intending to increase data processing effectiveness and avoid algorithmic errors [3]. This is often conducted using metaheuristic algorithms like genetic algorithms and particle swarm optimization [4]. Among these metaheuristic approaches, the Whale Optimization Algorithm (WOA) has emerged as a promising choice for feature selection. However, the WOA is susceptible to local optima trapping, a challenge in large datasets.

This study addresses the local optima problem in the WOA for feature selection in SFP by introducing the truncation selection method. Building on recent advancements in WOA variants [5], this research investigates the effectiveness of truncation selection within the context of WOA selection enhancement. This novel approach aims to improve WOA's performance in SFP, offering a potential solution to the local optima challenge.

In summary, this research aims to contribute to the field of software fault prediction by leveraging metaheuristic algorithms, specifically the WOA, in combination with the novel truncation selection method, building upon previous advancements to address local optima challenges and enhance the effectiveness of machine learning algorithms in software fault prediction.

2. Review of Related Work

A comprehensive review of the literature related to Software fault prediction (SFP), machine learning (ML)-based SFP, feature selection, and metaheuristic algorithms for software fault prediction is presented. This review aims to provide a foundational understanding of existing knowledge in this field to support the development and evaluation of the proposed methodology.

A study by [4] brings to the fore the traditional techniques used in SFP, encompassing software matrices, soft computing (SC), and machine learning (ML). While these approaches have significantly contributed to early fault prediction and the development of dependable software, they still have limitations in predicting certain types of faults. Additionally, they might be time-consuming, particularly when applied to complex software projects, leading to potentially diminished software testing effectiveness.

Some examples of ML-based SFP techniques include [6–10].

Feature selection (FS) has become a significant step in data mining, in general, and machine learning, in particular, since it helps to clean data by removing noisy, irrelevant, and redundant features [11]. A study by [12] developed a novel FS called evolving populations with mathematical diversification (FS-EPWMD), which uses arithmetic diversification among candidate solutions to avoid the local optimum. The guiding principle of populations evolving through crossover and mutation is the survival of the fittest. The results demonstrated that FS-EPWMD outperforms other models.

Swarm intelligence (SI) is a computational intelligence technique used to resolve complicated problems [13]. Swarm intelligence algorithms have demonstrated excellent performance in lowering the running time and addressing the FS problem. For instance, in order to solve the FS problem in the area of software fault prediction, ref. [14] proposed the island model to improve the BMFO. The EBMFO with the SVM classifier produced the best results overall. These findings show that the suggested model can be a useful predictor for the software fault issue.

3. Proposed Work

This section presents the research workflow and outlines the proposed methodology employed in this study, covering the enhancement of the algorithm and subsequent performance evaluation.

3.1. The Research Workflow

The proposed model works in four phases, namely, the literature review, methodology, implementation, and evaluation and results phase. The flow begins from the literature review down to the evaluation and results, and each phase is represented by its activities. Figure 1 represents the entire workflow of the model.

In the first phase, we conducted a comprehensive literature review, exploring 42 articles related to software fault prediction (SFP), ML-based SFP, feature selection (FS), and selection schemes. The second phase highlighted the main components of the proposed ML-based SFP model and the enhanced WOA. ML-based SFP handles prediction, while the enhanced WOA aims to enhance its performance. In the third phase, we implemented the ML-based SFP model using Google Colab and replicated baseline work in the same environment for comparison. To classify FS problems, four well-known classifiers were employed: support vector machine (SVM), decision tree (DT), linear discriminant analysis (LDA), and K-Nearest Neighbors (KNN). In the final phase, the proposed model's performance was evaluated using the following metrics: area under curve (AUC), precision, recall, F1 score, and accuracy. The cross-validation technique was also used to assess the performance of the model, where 80% of the dataset was used for training and 20% was used for testing.

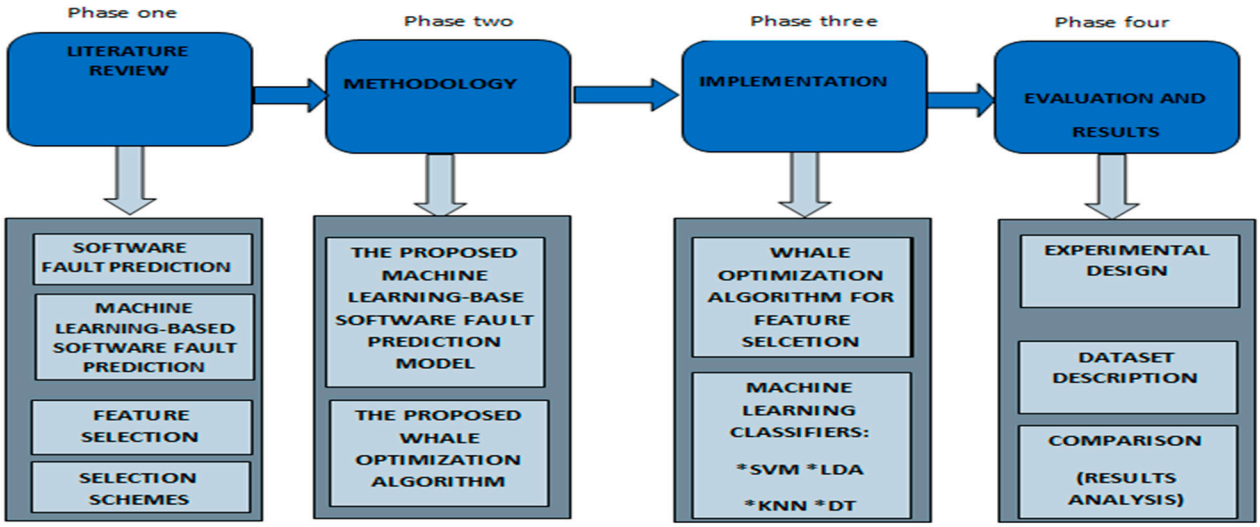

Figure 1. The research workflow.

3.2. The Proposed ML-Based SFP Model

This section details the research methodology for ML-based software fault prediction (SFP). Figure 2 presents the proposed model diagram, which works in five stages: data collection, data pre-processing, feature selection, machine learning classifiers, and evaluation. We presented the description of the PROMISE dataset in Table 1.

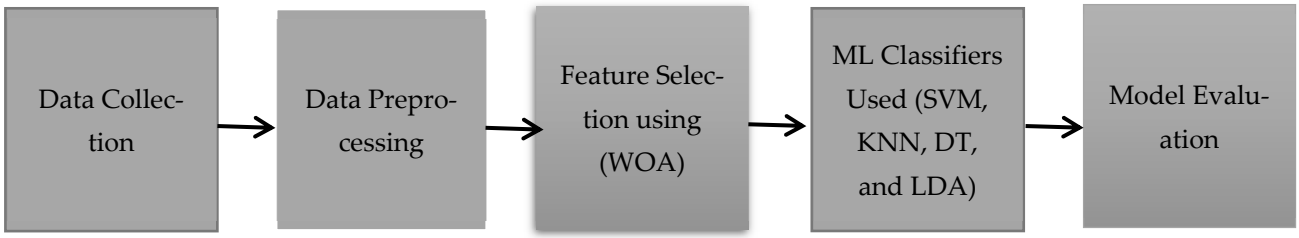

Figure 2. The proposed model.

Table 1. Description of the PROMISE Dataset.

Dataset	Version	# Instances	# Defective Instances	% Defective Instances
Ant	1.7	745	166	0.223
Camel	1.2	608	216	0.355
Camel	1.4	872	145	0.166
Camel	1.6	965	188	0.195
Jedit	3.2	272	90	0.331
Jedit	4.0	306	75	0.245
Jedit	4.1	312	79	0.253
Jedit	4.2	367	48	0.131
log4j	1.0	135	34	0.252
log4j	1.1	109	37	0.339
log4j	1.2	205	189	0.922
Lucene	2.0	195	91	0.467
Lucene	2.2	247	144	0.583
Lucene	2.4	340	203	0.597

Stage 1 involves gathering 14 datasets from the PROMISE dataset repository, with details provided in Table 1. In Stage 2, data pre-processing was used to stabilize the dataset into a form suitable for training and validation. Stage 3 employs the Whale Optimization Algorithm (WOA) for feature selection, with a focus on improving its selection scheme

using truncation selection. Stage 4 deploys four ML classifiers (DT, KNN, LDA, and SVM) to predict software faults, enhancing the WOA's performance in feature selection. In the final stage, we evaluated the model using the evaluation metrics mentioned above.

3.3. The Proposed Enhanced Whale Optimization Algorithm

This work employs the Whale Optimization Algorithm (WOA) and enhances it by incorporating truncation selection to improve its selection scheme, as illustrated in Figure 3. In Figure 3, we addressed the challenge of a stuck best solution in local optima, where we proposed a solution involving the combination of the Whale Optimization Algorithm (WOA), truncation selection, and single-point crossover approaches. This enhancement primarily focuses on improving the selection part of the WOA, where truncation selection is employed. In the truncation selection process, individuals are ranked based on their fitness values, and only the best-performing individuals are chosen as parents for the next generation. This selection is governed by a primary truncation selection parameter known as the TRS threshold, which can vary between 50% and 10%.

It determines the percentage of the population that will serve as parents. Individuals falling below this threshold are eliminated as they are considered unfit for reproduction. The truncation selection processes are indicated below:

- The population is sorted based on each individual's evaluation scores;
- The poorest-performing fraction of the population is removed;
- The eliminated individuals are replaced with individuals from the top-performing fraction, with each of the best individuals creating one offspring. These offspring subsequently replace one of the previously removed, lower-performing individuals.

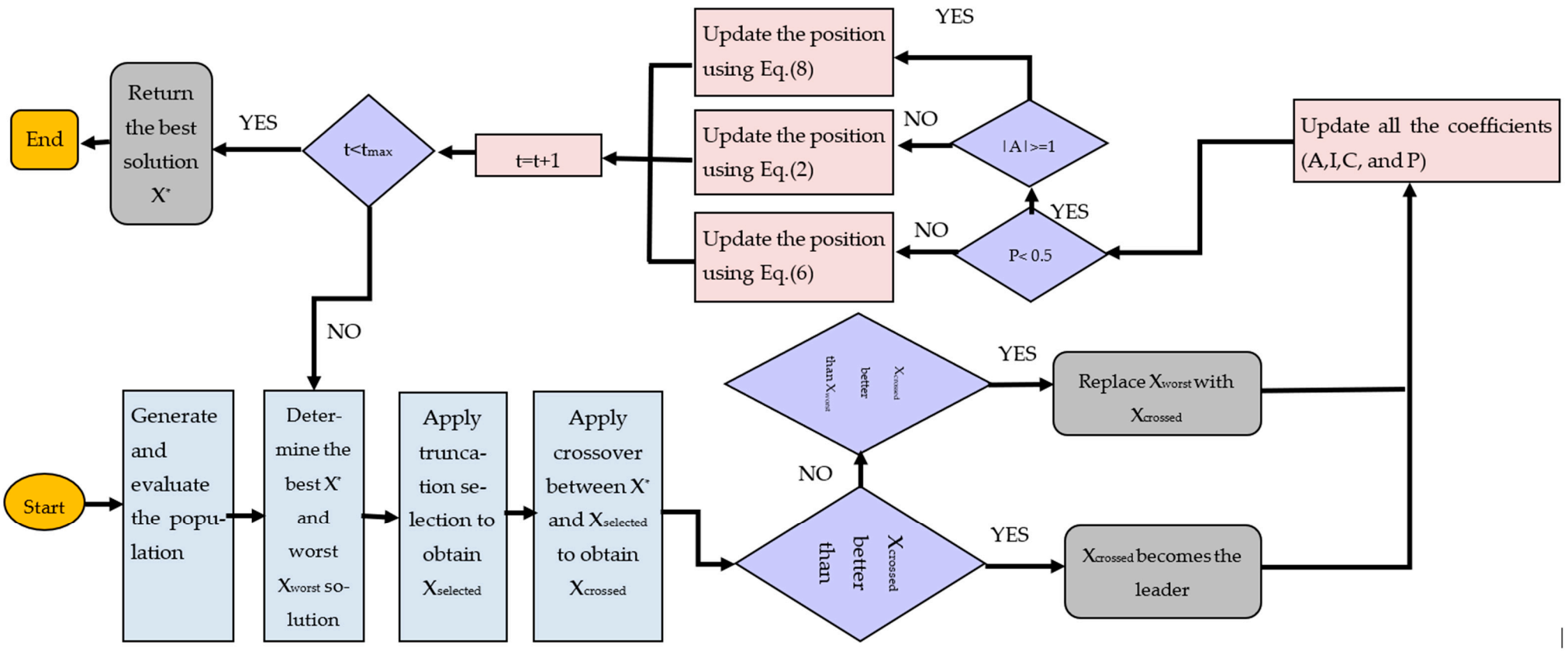

Figure 3. Flowchart of the proposed enhanced WOA.

4. Discussion of Results

This section is devoted to the presentation of the experimental results.

4.1. Implementation Environment

The work was implemented on the Google Colab environment using Python, Pandas, and Tensor Flow libraries.

4.2. Proposed Model Performance

The proposed enhanced WOA was evaluated by iteratively enhancing a population of candidate solutions using the truncation selection method, crossover, and coefficient update procedures. The results obtained from the experiments revealed promising outcomes, as shown in Table 2.

Table 2. Truncation-based WOA (TRBWOA) with SVM.

Datasets	Accuracy	Precision	Recall	F1 Score	AUC
Ant-1.7	0.839	0.650	0.533	0.519	0.687
Camel-1.2	0.706	0.600	0.102	0.239	0.604
Camel-1.4	0.817	0.04	0.05	0.561	0.493
Camel-1.6	0.877	0.433	0.224	0.440	0.605
Jedit-3.2	0.763	0.900	0.528	0.680	0.699
Jedit-4.0	0.922	0.800	0.386	0.421	0.734
Jedit-4.1	0.730	0.875	0.486	0.513	0.819
Jedit-4.2	0.978	0.0	0.500	0.500	0.892
Log4j-1.0	0..740	0.433	0.661	0.322	0.736
Log4j-1.1	0.829	0.725	0.625	0.625	0.705
Log4j-1.2	0.902	0.902	0.950	0.948	0.500
Lucene-2.0	0.767	0.300	0.873	0.518	0.676
Lucene-2.2	0.580	0.650	0.850	0.701	0.850
Lucene-2.4	0.806	0.588	0.753	0.740	0.849

In Table 2, Ant-1.7 performed best with an accuracy of 0.834, while Lucene-2.2 scored the lowest at 0.534. Log4j-1.2 had the highest precision (0.789), and Log4j-1.0 had the highest recall (0.727).

Log4j-1.0 also achieved the highest F1 score (0.643) and AUC (0.790). Conversely, Camel-1.6 consistently had lower scores across these metrics.

In Table 3, we summarized the dataset performance. Ant-1.7 had the best accuracy (0.832), while Camel-1.6 scored the lowest (0.878). Camel-1.4 achieved perfect precision (1.0), while Camel-1.6 had the lowest accuracy (0.333). Camel-1.2 had the highest recall (0.504), and Jedit-4.2 had the highest F1 score (0.500) and AUC (0.863). Camel-1.4 had the lowest AUC (0.517) and Camel-1.6 had the lowest F1 score (0.244)

Table 3. Truncation-based WOA (TRBWOA) with KNN.

Datasets	Accuracy	Precision	Recall	F1 Score	AUC
Ant-1.7	0.832	0.581	0.600	0.533	0.745
Camel-1.2	0.715	0.656	0.504	0.275	0.623
Camel-1.4	0.834	1.0	0.333	0.365	0.517
Camel-1.6	0.878	0.333	0.438	0.244	0.605
Jedit-3.2	0.781	0.946	0.523	0.647	0.863
Jedit-4.0	0.874	0.500	0.387	0.463	0.850
Jedit-4.1	0.777	0.827	0.481	0.540	0.732
Jedit-4.2	0.846	1.0	0.450	0.500	0.801
Log4j-1.0	0..919	0.767	0.725	0.400	0.766
Log4j-1.1	0.815	0.950	0.777	0.867	0.625
Log4j-1.2	0.864	0.902	0.433	0.949	0.743
Lucene-2.0	0.902	0.444	0.876	0.381	0.812

Table 3. *Cont.*

Datasets	Accuracy	Precision	Recall	F1 Score	AUC
Lucene-2.2	0.766	0.756	0.629	0.812	0.600
Lucene-2.4	0.618	0.657	0.753	0.690	0.749

4.3. Results Comparison

Regarding AUC, Table 4 shows that TRBWOA performed better than TBWOA and all other variations when decision tree was applied. The TRBWOA performed remarkably well in datasets like ant-1.7 with 0.803, while the Random-Based Whale Optimization Algorithm (RBWOA) had the worst performance in log4j-1.2 with 0.486. Table 4 shows the results of the comparison of the models.

Table 4. Comparison of results of the WOA implemented with different selection schemes with DT classifier in terms of AUC.

Datasets	LRBWOA	RBWOA	PBWOA	TBWOA	SUSBWOA	TRWOA
Ant-1.7	0.690	0.687	0.664	0.688	0.664	0.987
Camel-1.2	0.635	0.609	0.604	0.606	0.612	0.608
Camel-1.4	0.531	0.585	0.582	0.587	0.589	0.555
Camel-1.6	0.593	0.575	0.575	0.567	0.576	0.679
Jedit-3.2	0.803	0.744	0.735	0.736	0.722	0.955
Jedit-4.0	0.569	0.560	0.571	0.550	0.567	0.655
Jedit-4.1	0.569	0.641	0.634	0.625	0.620	0.855
Jedit-4.2	0.782	0.718	0.701	0.725	0.730	0.665
Log4j-1.0	0.777	0.644	0.640	0.653	0.657	0.638
Log4j-1.1	0.562	0.605	0.713	0.704	0.712	0.777
Log4j-1.2	0.597	0.486	0.643	0.660	0.627	0.925
Lucene-2.0	0.504	0.560	0.499	0.496	0.762	0.751
Lucene-2.2	0.533	0.650	0.528	0.551	0.507	0.654
Lucene-2.4	0.642	0.634	0.615	0.634	0.536	0.761

Regarding AUC, Table 5 shows that TRBWOA outperformed all other variations of the WOA, including the TBWOA, when KNN was applied. The TRBWOA performed well in datasets like Jedit-3.2 with 0.863, while the linear ranked-based Whale Optimization Algorithm (LRBWOA) had the worst performance in camel-1.6 with 0.47. Figure 4 shows the graphical presentation of the results compared.

Table 5. Comparison of results of the WOA implemented with different selection schemes with KNN classifier in terms of AUC.

Datasets	LRBWOA	RBWOA	PBWOA	TBWOA	SUSBWOA	TRBWOA
Ant-1.7	0.657	0.657	0.682	0.691	0.704	0.745
Camel-1.2	0.524	0.582	0.519	0.525	0.517	0.623
Camel-1.4	0.555	0.499	0.511	0.517	0.511	0.517
Camel-1.6	0.474	0.556	0.496	0.505	0.505	0.605
Jedit-3.2	0.703	0.780	0.735	0.756	0.719	0.863
Jedit-4.0	0.544	0.569	0.552	0.634	0.557	0.850
Jedit-4.1	0.619	0.654	0.650	0.500	0.647	0.732
Jedit-4.2	0.679	0.804	0.662	0.669	0.689	0.801
Log4j-1.0	0.488	0.761	0.607	0.679	0.604	0.766
Log4j-1.1	0.660	0.633	0.691	0.516	0.677	0.625
Log4j-1.2	0.625	0.500	0.520	0.516	0.524	0.743
Lucene-2.0	0.546	0.601	0.545	0.531	0.551	0.812
Lucene-2.2	0.640	0.476	0.593	0.715	0.586	0.600
Lucene-2.4	0.639	0.619	0.573	0.792	0.583	0.749

Figure 4. Comparison of results of the WOA implemented with different selection schemes with KNN classifier in terms of AUC.

5. Conclusions

Remarkably, the proposed work demonstrated significant advancements over the previous work across all evaluated metrics and datasets. The results showcased the superior performance of the proposed work, consistently outperforming the previous work in terms of various evaluation measures. These findings affirm the efficiency and effectiveness of the proposed approach in addressing the research objectives and achieving improved outcomes.

Author Contributions: Contribution: Conceptualization, H.A. and L.Y.; methodology, H.A.; software, H.A.; validation, H.A., K.U. and R.A.; resources, H.A. and K.M., supervision, K.U. and L.Y.; project administration, K.U. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data were obtained from datasets gathered from the PROMISE dataset repository [http://promise.site.uottawa.ca/SERepository/].

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

References

1. Rathore, S.S.; Kumar, S. A study on software fault prediction techniques. *Artif. Intell. Rev.* **2019**, *51*, 255–327. [CrossRef]
2. Singh, P.D.; Chug, A. Software defect prediction analysis using machine learning algorithms. In Proceedings of the 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence, Noida, India, 12–13 January 2017.
3. Cai, J.; Luo, J.; Wang, S.; Yang, S. Feature selection in machine learning: A new perspective. *Neurocomputing* **2018**, *300*, 70–79. [CrossRef]
4. Hassouneh, Y.; Turabieh, H.; Thaher, T.; Tumar, I.; Chantar, H.; Too, J. Boosted whale optimization algorithm with natural selection operators for software fault prediction. *IEEE Access* **2021**, *9*, 14239–14258. [CrossRef]
5. Heidari, A.A.; Aljarah, I.; Faris, H.; Chen, H.; Luo, J.; Mirjalili, S. An enhanced associative learning-based exploratory whale optimizer for global optimization. *Neural Comput. Appl.* **2020**, *32*, 5185–5211. [CrossRef]
6. Bowes, D.; Hall, T.; Petrić, J. Software defect prediction: Do different classifiers find the same defects? *Softw. Qual. J.* **2018**, *26*, 525–552. [CrossRef]
7. Pandey, S.K.; Mishra, R.B.; Tripathi, A.K. Machine learning based methods for software fault prediction: A survey. *Expert Syst. Appl.* **2021**, *172*, 114595. [CrossRef]
8. Suryadi, A. Integration of feature selection with data level approach for software defect prediction. *Sink. J. Dan Penelit. Tek. Inform.* **2019**, *4*, 51–57. [CrossRef]
9. Bahaweres, R.B.; Suroso, A.I.; Hutomo, A.W.; Solihin, I.P.; Hermadi, I.; Arkeman, Y. Tackling feature selection problems with genetic algorithms in software defect prediction for optimization. In Proceedings of the 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), Jakarta, Indonesia, 19–20 November 2020.

10. Alkhasawneh, M.S. Software Defect Prediction through Neural Network and Feature Selections. *Appl. Comput. Intell. Soft Comput.* **2022**, *2022*, 2581832. [CrossRef]
11. Chantar, H.; Mafarja, M.; Alsawalqah, H.; Heidari, A.A.; Aljarah, I.; Faris, H. Feature selection using binary grey wolf optimizer with elite-based crossover for Arabic text classification. *Neural Comput. Appl.* **2020**, *32*, 12201–12220. [CrossRef]
12. Goyal, S. Software fault prediction using evolving populations with mathematical diversification. *Soft Comput.* **2022**, *26*, 13999–14020. [CrossRef]
13. Shrivastava, D.; Sanyal, S.; Maji, A.K.; Kandar, D. Bone cancer detection using machine learning techniques. In *Smart Healthcare for Disease Diagnosis and Prevention*; Elsevier: Amsterdam, The Netherlands, 2020; pp. 175–183.
14. 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]