Ensemble Learning Model based Software Defect Detection Using Bio-Inspired Optimized Features

Kavita Chourasia[1], Harsh Mathur[2]

[1]Phd Scholar, Dept. of Computer Science, RabindraNath Tagore University, Bhopal, MP, India, Email:kavichourasia@gmail.com
[2]Associate Professor, Dept. of Computer Science, RabindraNath Tagore University, Bhopal, MP, India, Email: harsh.mathur@aisectuniversity.ac.in

Received: 18.04.2024 Revised : 13.05.2024 Accepted: 24.05.2024

ABSTRACT
software development, defects are an inherent aspect of the process and can arise at any phase, including requirements gathering, coding, or testing. These defects, whether they emerge during initial planning or late in the development cycle, pose significant challenges and can affect the final product's functionality and performance. Implementing software defect prediction (SDP) techniques is crucial in managing these challenges effectively. SDP can play a vital role in reducing associated costs by identifying potential issues early, thus allowing for more efficient allocation of resources and targeted testing efforts. This paper has proposed a ensemble learning model to identify the defects in the software at early stage of development. In order to improve the software defect prediction accuracy feature optimization was done by using bioinspird algorithm. Experiment was done on real dataset having data of multiple software. Results shows that use of ensemble model with optimize feature has increases the detection accuracy.

Keywords: Deep Learning, Software Defect Detection Genetic Algorithm, Feature optimization.

1. INTRODUCTION

In the ever-evolving field of software development, striving for flawless and efficient software remains a constant challenge. Despite best efforts, software defects are an inevitable aspect of this intricate process. The topic of Software Defect Prediction (SDP) has garnered significant attention in academic literature, with various methods and techniques being developed to predict defects across different datasets. Software Defect Detection (SDD) is a crucial yet often underappreciated component of the development lifecycle. Even minor defects can lead to impaired software functionality, reduced system stability, or potentially catastrophic system crashes, all of which can significantly detract from the user experience [1]. Historically, SDD has been a labor-intensive and time-consuming task, demanding substantial effort from developers and quality assurance teams. Consequently, the pursuit of intelligent SDD has been a longstanding area of focus in software engineering research, aiming to streamline the detection process and improve the overall quality of software products.

Research into dimensionality reduction techniques that incorporate global features, local features, and kernel-based feature subsets to address nonlinearity in software defect prediction remains limited. The presence of nonlinear relationships and high dimensionality in defect datasets can significantly affect the performance of predictive models. To address this issue, we propose a novel software defect prediction algorithm that utilizes learnable three-line hybrid feature fusion, inspired by the principles of three-line hybrid rice breeding. This approach aims to enhance the prediction process by effectively integrating diverse feature sets. In our prior work, we introduced an adaptive variable sparrow search algorithm (AVSSA), which demonstrated exceptional search efficiency in experimental evaluations (Tang et al., 2023).

Early detection of product errors enables software to be adapted to various conditions and enhances resource utilization. In the realm of computer programming, testing and debugging can be prohibitively costly, requiring extensive resources [12]. Effective defect prediction can reduce testing expenses and contribute to improved software quality [7, 8]. Prediction strategies range from traditional product measurements to advanced machine learning (ML) and Software Computing (SC) techniques [9]. Artificial intelligence (AI) systems play a crucial role in predicting software errors [13]. Various feature selection methods have been employed to identify optimal subsets of features for more accurate predictions, as irrelevant or redundant features can skew the performance of classification models [10, 14].

The remainder of this paper is organized as follows: Section 2 provides an overview of the various defect models that have been proposed for software. This section offers a comprehensive review of these models, highlighting their methodologies and applications. In Section 3, we present a detailed description of the SDDELBOF model. This includes an in-depth explanation accompanied by a block diagram that illustrates the model's architecture and operational components. Moving to Section 4, we outline the experimental setup and procedures employed in our study. This section details the methodology used to test and validate the model. Finally, Section 5 presents the results of the experiments, discusses their implications, and draws conclusions based on the findings.

2. RELATED WORK

R. Haque and colleagues (2024) introduced an advanced method called heterogeneous cross-project defect prediction (HCDP), leveraging encoder networks alongside a transfer learning (ENTL) model to forecast software defects [18]. This methodology utilized encoder networks (ENs) to distill crucial features from both the source and target datasets. To mitigate the challenge of negative transfer in transfer learning (TL), an augmented dataset, enriched with pseudolabels and data from the source dataset, was employed. The model was initially trained using a single dataset and subsequently tested on sixteen datasets extracted from four public projects. The study also included a comparative analysis with conventional techniques, employing cost-sensitive learning (CL) to manage issues related to class imbalance.

In a parallel effort, Y. Al-Smadi and co-authors (2023) proposed a novel approach for software defect prediction by deploying 11 machine learning (ML) techniques on 12 varied datasets [19]. They incorporated four diverse meta-heuristic algorithms—particle swarm optimization (PSO), genetic algorithm (GA), harmony search (HS), and ant colony optimization (ACO)—to identify the optimal set of features. To address the issue of data imbalance, they applied the synthetic minority oversampling technique (SMOTE). Furthermore, they utilized the Shapley additive explanation (SAE) framework to pinpoint the most influential features.

A. Wang and colleagues (2023) developed an innovative approach called Federal Prototype Learning with Prototype Averaging (FPLPA), combining federated learning (FL) and prototype learning (PL) to predict heterogeneous software defects [20]. This method involved the use of the one-sided selection (OSS) algorithm to filter out noise from local training datasets, ensuring that the data used was clean and relevant. To select the most informative features, they applied the Chi-Squares Test algorithm, which identified the optimal subset of features necessary for accurate defect prediction. The team then introduced the Convolution Prototype Network (CPN) model, designed to create local prototypes that were more robust to heterogeneous data than traditional convolutional neural networks (CNNs). This approach was particularly effective in mitigating the impact of class imbalances within the software data, a common challenge in defect prediction.

In the FPLPA framework, these prototypes served as the main communication medium between individual clients and the central server, with the local prototypes being developed in a way that ensures they cannot be reversed, thereby protecting privacy during the communication process. The final step involved refining the model by using the loss of both local and global prototypes as a form of regularization, ensuring that the model remained accurate and effective across different data distributions. The proposed method was rigorously tested using the AEEEM, NASA, and Relink datasets. Simulation results demonstrated its superiority over traditional methods, showcasing its effectiveness in accurately predicting software defects.

In a related study, S. H. A. Hamid and colleagues explored the use of Deep Q-Networks (DQN) in software defect prediction, emphasizing the reduction of false positives while enhancing overall prediction performance. During the training phase, the model was guided by a reward policy designed to minimize incorrect predictions, thereby improving its ability to identify defect-prone software components accurately. This approach highlights the potential of reinforcement learning techniques, like DQN, in refining defect prediction models and achieving better predictive accuracy.

T. Tahir and colleagues conducted a systematic mapping study (SMS) on secondary studies, specifically literature reviews, related to defect prediction. Their goal was to identify the most frequently used datasets, project attributes, metrics employed as predictors, and supervised learning methods utilized for training the data. Following this, they performed an empirical study focused on predicting defect density using cross-project data. They gathered data from 760 projects in the International Software Benchmarking Standards Group (ISBSG) dataset, version 11, which included both defect reports and functional size attributes. The team trained their prediction models using three approaches: i) the complete set of project attributes, ii) individual attributes, and iii) various subsets of attributes. They employed both classification and regression machine learning techniques to evaluate the effectiveness of

these models.

S. Kwon and colleagues proposed a function-level just-in-time (JIT) software defect prediction (SDP) model. This model addresses the limitation of allocating limited testing resources by prioritizing defect-prone functions. They utilized a pre-trained transformer-based deep learning model, trained on a large corpus of code snippets, to predict the likelihood of defects in modified functions at the commit level. The fine-tuned version of this pre-trained model can assess defect proneness for specific code changes. They evaluated the performance of three popular pre-trained models—CodeBERT, GraphCodeBERT, and UniXCoder—within both within-project and cross-project environments, particularly focusing on edge-cloud systems.

S. Kaliraj and colleagues tackled two significant challenges in software fault prediction. First, they addressed the issue of class imbalance, which can greatly affect the accuracy of predictive models. Through extensive experimentation with various classifiers across diverse datasets from different software projects, they highlighted how classifier performance varies and underscored the importance of addressing class imbalance for reliable predictions. Second, they evaluated the reliability of cross-project prediction, investigating how well predictive models trained on one project can generalize to others. Their study demonstrated that using datasets with characteristics similar to the target project is crucial for achieving accurate and reliable cross-project predictions.

3. PROPOSED METHODOLOGY

This study introduces a comprehensive methodology for detecting software defects during the final stages of development. The proposed approach, known as SDDELBOF (Software Defect Detection by Ensemble Learning and Bio-Inspired Optimized Features), initiates with a thorough process of dataset preparation and feature clustering. These preliminary steps are critical as they set the foundation for effective defect detection by ensuring that the data is well-structured and relevant features are identified and grouped accordingly. The methodology employs ensemble learning techniques combined with bio-inspired optimization algorithms to enhance the accuracy of defect detection. Figure 1 provides a detailed block diagram of the software defect detection model, outlining each component of the system and its role in the overall process. This diagram visually represents the workflow from data preprocessing to the final detection and classification of software defects, highlighting the intricate steps involved in the methodology.

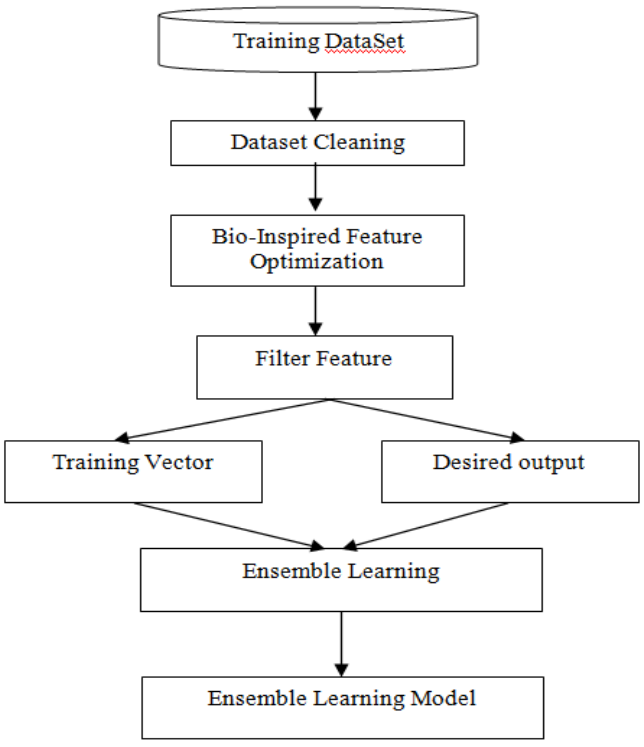

Fig 1. SDDELBOF training model diagram.

Raw Dataset Cleaning

To adapt the original unstructured data to the required format, the cleaning process involved restructuring the dataset into specific rows and columns. This process included removing irrelevant elements such as software versions, class names, and function names. Missing data or cells with null values were addressed by using either the column's average or a zero count [11]. The data's numeric range varied, with some metrics, like 'Data Access Metrics' and 'Cohesion across Methods,' falling between zero and one, while others, such as 'Inheritance Depth' and 'Measure of Aggregation,' were in the hundreds. For instance, the line count of code could reach into the thousands. Each column's values were individually normalized using Equation 1 [12].

$ND_{i,j}$ = Normalizeddata(SRD)----------Eq. 1

The values in columns PC are normalized from zero to one using Eq. 1.

Feature optimization

This work uses two of bio-inspired algorithms for the feature optimization. First one is Elephant herd [13, 14] and other was artificial immune algorithm [15, 16]. Both of these algorithm have some common steps of feature clustering. Features were cluster into two group first was elected features fir for training of ensemble model and other was rejected features not fit for training. Each of those common steps (generate population, fitness, crossover, population update were detailed in the section:

Generate Populationpreprocessed features were randomly classified into selected and rejected class. A binary vector represent the label 1 or 0 for each feature as per the selection or rejection class. For randomization Gaussian function was used.Each vector is terms as elephant in elephant herd algorithm [14]. Similarly each vector is terms as antibody in artificial immune algorithm [16]. Collection of vector is Pv population in the algorithm, hence this work have Pm number of population and for Fn number of features.

Pv←Generate_vectors(Pm, Fn)

Fitness Since each of vector was randomly generated hence fitness value need to be evaluate for the same. As features were used for the training of model so temporary model was train by the selected feature vector. So each of algorithm uses common fitness function of training of model and test the trained model to get the accuracy of detection. Detection accuracy is the fitness value.

Pvf←Fitness(Pv, ND) ----------Eq. 3

Crossover Vectors need updation as per the fitness value. So each of elephant or antibody get update by the best feature vector. Modification was done on the basis of changing the feature value status from selection to rejection or rejction to selection as per best feature vector of current iteration.

Pv←Crossover_updates(Pv, Pvf)

Update Population Each of bioinspired algorithm generate new feature set by crossover operation. This step balance population by removing low fitness feature vector either parent or child. This step also check that maximum iteration count to get the final updated population and get cluster feature set.

Training of Mathematical Model

A bootstrap ensemble combines multiple learning model, each built from a bootstrap sample of the training data, to improve the accuracy and robustness of predictions.

Bootstrap Sampling: This involves randomly selecting samples with replacement from the original dataset to create multiple subsets [17]. Each subset may have duplicate entries from the original data. This technique helps in creating a diverse set of training samples.

Ensemble Learning: This involves combining the predictions of multiple models to obtain a more accurate and stable prediction. In the case of a Random Forest, it aggregates the results of multiple decision trees to make a final prediction.

Proposed SDDELBOF **Algorithm**

Input: SDRD // Software Defect Raw Dataset

Output: ESDDM // Ensemble Software Defect Detection Model

1. PD←Dataset_Cleaning(SDRD)
2. ND=Normalizeddata(SRD)
3. Pv←Generate_vectors(Pm, Fn)
4. Loop 1:T

5. Pvf←Fitness(Pv, ND)
6. Pv←Crossover_updates(Pv, Pvf)
7. Pv←UpdateVectorPopulation(Pv, Pvf)
8. End Loop
9. Bv←Fitness(Pv, ND) // Best Vector
10. Fs←Cluster(ND, Bv)
11. ESDDM ←TrainEnsembleModel(Fs, ND)

The raw dataset undergoes preprocessing to prepare it for analysis in the SDDELBOF model, which is designed for detecting software bugs. This preprocessing stage involves cleaning the data, handling missing values, and transforming the dataset into a suitable format for further analysis. After preprocessing, the model employs a bio-inspired optimization approach to select the most relevant and effective features for detecting software bugs. This feature selection is crucial, as it helps to enhance the model's accuracy and efficiency by focusing on the most informative aspects of the data. Additionally, the model utilizes ensemble learning techniques, which combine multiple machine learning algorithms to improve predictive performance and robustness. Once the model is trained with the optimized features, it becomes capable of predicting potential defects in the software, providing a powerful tool for identifying and addressing bugs during the development process.

4. EXPERIMENT AND RESULTS

The implementation of the model involved utilizing a MATLAB program on a computer equipped with a 4 GB RAM and an i6 generation CPU. To assess the performance of the model, metrics including precision, recall, f-measure, and accuracy were employed for comparative analysis.Comparisonof proposed model was done with Hellinger Net [18] and [19]. It was found that use of elephant herd optimization perform better as compared to artificial immune genetic algorithm, hence comparison was doen with elephant herd optimized features.

Dataset

In order to conduct the experiments, the IC-DePress dataset served as the primary dataset for the program. The evaluation of the proposed model spanned across six separate projects, encompassing a total of 13,522 sessions within the dataset. A comparison of the proposed model was conducted using software fault detection methodologies as outlined in [20].

Results

Table 1. Accuracy based Software Defect Detection models comparison.

Software	Hellinger Net	SDDEHF	SDDELBOF
SoftCamel	83.12	91	98.43
SoftIVY	83.55	91.45	97.64
SoftJEdit	87.45	0.9275	99.19
SoftLicene	73.34	82.28	98.51
SoftPOI	69.04	0.8707	96.91

Table 1 shows that SDDELBOF has high defect detection accuracy as compared to existing models. Bioinspired Optimized feature set were used for the training of ensemble model has increases the detection accuracy of the software defects. Accuracy values were improved by 9.401% as compared to SDDEHE model.

Table 2. Precision based Software Defect Detection models comparison.

Software	Hellinger Net	SDDEHF	SDDELBOF
S_Camel	0.9556	0.9616	0.9973
S_IVY	0.9974	0.9769	0.9949
S_JEdit	0.9825	0.9812	0.9969
S_Licene	0.855	0.9135	0.9975
S_POI	0.8898	0.9139	0.9862

Precision values compared in the table 2 shows that use of ensemble model for the software defect

detection in SDDELBOF has increased precision values as compared to existing models. Further it was found that clustered features with normalization has increase the learning precision value by 4.53% as compared to SDDEHF model.

Table 3. Recall based Software Defect Detection models comparison.

Software	Hellinger Net	SDDEHF	SDDELBOF
S_Camel	0.8311	0.9322	0.9841
S_IVY	0.8351	0.9248	0.9773
S_JEdit	0.8736	0.9378	0.9938
S_Licene	0.7417	0.831	0.98
S_POI	0.6905	0.9	0.9695

Table 3 shows recall values for the different softwares and it was found that SDDELBOF has improved the recall value 19.01% as compared to Hellinger Net model.

Table 4. F-measure based Software Defect Detection models comparison.

Software	Hellinger Net	SDDEHF	SDDELBOF
S_Camel	0.9078	0.9467	0.9906
S_IVY	0.9102	0.95	0.986
S_JEdit	0.9323	0.959	0.9953
S_Licene	0.8156	0.8703	0.9887
S_POI	0.8163	0.9069	0.9778

Table 4 shows that SDDELBOF has high defect detection f-meausre as compared to existing models. Bioinspired Optimized feature set were used for the training of ensemble model has increases the detection f-measure of the software defects. F-measure values were improved by 6.18% as compared to SDDEHE model.

Table 5. Reliability based Software Defect Detection models comparison.

Software	Hellinger Net	SDDEHF	SDDELBOF
S_Camel	0.9469	0.9469	0.9468
S_IVY	0.3951	0.4551	0.3969
S_JEdit	0.9271	0.927	0.927
S_Licene	0.598	0.6744	0.5968
S_POI	0.607	0.6418	0.6055

Reliability values compared in the table 5 shows that use of ensemble model for the software defect detection in SDDELBOF has increased reliability values as compared to existing models. Further it was found that clustered features with normalization has increase the learning efficiency of model.

5. CONCLUSIONS

This paper has developed a model that accurately identify software defects with less number of features. Whole work focuses on the feature optimization and learning of feature for defect prediction. Feature optimization was done by bioinspired algorithm elephant herd. Learning of optimized features were used for the training of ensemble model. Experiment was done on different dataset and compared with various model. Result shows that that use of ensemble model for the software defect detection in SDDELBOF has increased precision values by 4.53%,. Further it was found that clustered features with normalization has increase the learning detection accuracy by 9.401% .In future researcher can develop a unsupervised model.

REFERENCES

[1] Iqbal A, Aftab S, Ali U, Nawaz Z, Sana L, Ahmad M, Husen A (2019) Performance analysis of machine learning techniques on software defect prediction using NASA datasets. Int J Adv Comput Sci Appl 10(5):300–308

[2] Lamba T, Mishra AK (2019) Optimal machine learning model for software defect prediction. Int J Intell Syst Appl 10(2):36 Google Scholar

[3] Gupta A, Sharma M, Srivastava A (2023) A Novel Dimensionality Reduction-based Software Bug Prediction using a Bat-Inspired Algorithm. In 2023 13th International Conference on Cloud Computing, Data Science & Engineering (Confluence) (pp. 278–285). IEEE.
[4] R. Haque, A. Ali, S. McClean, I. Cleland and J. Noppen, "Heterogeneous Cross-Project Defect Prediction Using Encoder Networks and Transfer Learning," in IEEE Access, vol. 12, pp. 409-419, 2024,
[5] Y. Al-Smadi, M. Eshtay and A. A. Abd El-Aziz, "Reliable prediction of software defects using Shapley interpretable machine learning models", Egyptian Informatics Journal, vol. 24, no. 3, pp. 386-394.
[6] A. Wang, L. Yang, H. Wu and Y. Iwahori, "Heterogeneous Defect Prediction Based on Federated Prototype Learning," in IEEE Access, vol. 11, pp.
[7] S. H. A. Hamid, , T. Tahir et al.,l, A. A. Sani and N. N. M. Daud, "Toward Reduction in False Positives Just-In-Time Software Defect Prediction Using Deep Reinforcement Learning," in IEEE Access, vol. 12, pp. 47568-47580, 2024.
[8] T. Tahir et al., "Early Software Defects Density Prediction: Training the International Software Benchmarking Cross Projects Data Using Supervised Learning," in IEEE Access, vol. 11, pp. 141965-141986, 2023
[9] S. Kwon, S. Lee, D. Ryu and J. Baik, "Pre-Trained Model-Based Software Defect Prediction for Edge-Cloud Systems," in Journal of Web Engineering, vol. 22, no. 2, pp. 255-278, March 2023.
[10] S. Kaliraj, A. M. Kishoore and V. Sivakumar, "Software Fault Prediction Using Cross-Project Analysis: A Study on Class Imbalance and Model Generalization," in IEEE Access, vol. 12, pp. 64212-64227, 2024
[11] Karpagalingam Thirumoorthy, Jerold John Britto J. "A feature selection model for software defect prediction using binary Rao optimization algorithm" Applied Soft Computing, Volume 131, 2022.
[12] Balogun, A.O.; Basri, S.; Capretz, L.F.; Mahamad, S.; Imam, A.A.; Almomani, M.A.; Adeyemo, V.E.; Alazzawi, A.K.; Bajeh, A.O.; Kumar, G. Software Defect Prediction Using Wrapper Feature Selection Based on Dynamic Re-Ranking Strategy. Symmetry 2021, 13, 2166.
[13] Monalisa Nayak, Soumya Das, Urmila Bhanja, Manas Ranjan Senapati, Elephant herding optimization technique based neural network for cancer prediction, Informatics in Medicine Unlocked, Volume 21, 2020.
[14] Wei Li, Gai-Ge Wang, Amir H. Alavi. "Learning-based elephant herding optimization algorithm for solving numerical optimization problems", Knowledge-Based Systems, Volume 195, 2020.
[15] Ying Tan, "Artificial Immune System," in Artificial Immune System: Applications in Computer Security , IEEE, 2016, pp.1-25.
[16] I F Astachova et al. "The application of artificial immune system to solve recognition problems". 2019 J. Phys.: Conf. Ser. 1203 012036.
[17] Naderalvojoud B, Hernandez-Boussard T. Improving machine learning with ensemble learning on observational healthcare data. AMIA Annu Symp Proc. 2024 Jan 11;2023:521-529.
[18] Tanujit Chakraborty and Ashis Kumar Chakraborty. "Hellinger Net: A Hybrid Imbalance Learning Model to Improve Software Defect Prediction". IEEE Transactions On Reliability, 2020.
[19] Kavita Chourasia, Dr. Harsh Mathur. "Elephant Herd Optimization for Software Defect Detection using Spiking Network". Conference 2024.
[20] Lech Madeyski, Marian Jureczko, "Which Process Metrics Can Significantly Improve Defect Prediction Models? An Empirical Study.", Software Quality Journal, vol. 23, no. 3, pp. 393–422, 2015.