Predicting software defects with swarm-intelligence-based machine learning algorithm for improved process quality

Rajesh Gupta a | Avinash Rajkumar b | Beemkumar N c

a Sanskriti University, Mathura, Uttar Pradesh, India, Pro Chancellor, Department of Management.
b Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India, Professor, Teerthanker Mahaveer Institute of Management and Technology.
c JAIN (Deemed-to-be University), Bangalore, India, Professor, Department of Mechanical Engineering.

Abstract The quality and effectiveness of software systems may be significantly impacted by Software Defects (SD). Therefore, enhancing process quality is essential for controlling and minimizing the incidence of faults. Implementing reliable Software Development (SDe) processes and best practices is one way to do this information. SD, commonly referred to as software bugs or software mistakes, are defects or errors that occur in computer programs and cause them to act up or create unintended outcomes. These vulnerabilities may appear for several causes, including programming mistakes, poor design choices, or issues with the SDe cycle. The prediction of software problems based on Machine Learning (ML) using an Enhanced Artificial Neural Network (E-ANN) is implemented in this study to increase software quality and testing effectiveness. Particle swarm optimization and grey wolf optimization, these two algorithms named grey wolf swarm optimization algorithm (GWSA), are combined to recognize the corresponding compensation of the methods following the respective benefits and drawbacks. The hybrid algorithm-based model to the conventional hyperparameter optimization strategy and a single swarm intelligence algorithm's investigation of investigational consequences from six data sets shows that the hybrid algorithm-based model has high and enhanced indicators. Processed by the autoencoder, the model's performance has also improved.

Keywords: software defects, E-ANN, GWSA, SDP

1. Introduction

Software reliability model-based Software Defect Prediction (SDP) is critical to assessing software superiority. Several primary duties make up the issue. The initial objective is to determine the suitable software reliability model's estimation parameters that provide the most significant match to software failure data. The second is to predict the time when SD will occur. An SD, sometimes called a software bug or software mistake, is a weakness or malfunction in a computer program that makes it act unexpectedly or provide inaccurate results. Numerous things, including programming mistakes, design faults, or issues with the SDe process, might lead to these defects (Yang et al 2021). In the process of building embedded software, one of the components considered to be among the most complex and costly is the process of locating and fixing defects. Measuring and attaining high standards is challenging, especially in automobile-embedded systems. This is primarily due to the complex infrastructure and its scope, cost, and time restrictions. Even so, achieving high standards for product quality and dependability is essential. Software testing requires the same time, money, infrastructure, and experience as software development. While developing safety-critical software systems, expenses and efforts rise. Consequently, every industry with significant SDe expenditures must have a solid testing plan. The software sector is expanding rapidly and technologically. Therefore, predicting software dependability is crucial in the SDe process (Thota et al 2020).

SD programmers often utilize debugging tools and procedures to find and fix coding issues. To identify flaws early on and stop them from becoming issues later, testing is a crucial component of the SDe process. Additionally, procedures for code review and software quality assurance may aid in lowering the possibility that errors would ever arise. Identification and reporting of SD are necessary for effective management of them. To find flaws, developers, and testers utilize a variety of methodologies, including manual testing, automated testing, code reviews, and user input. It’s essential to have a well-organized defect reporting procedure to guarantee accurate and valuable information (Li et al 2020). The practice of identifying components of a software system that could have flaws is known as Defect Prediction (DP) in software. In addition to easing up on the maintenance effort, this results in lower labour expenses throughout development. At first, the frameworks utilized in DP were developed using statistical methods. Still, learning techniques must be used in building DP models for the model to be intelligent, that is, capable of modifying evolving data so that as the creation process matures,
the DP model also matures. Priorities should be determined by considering a defect’s severity, frequency of recurrence, possible business effect, and user impact. Critical flaws should be fixed as soon as possible if they threaten the system’s security, stability, or fundamental functioning (Son et al 2019).

Digital software systems are becoming more sophisticated, and the resulting software applications often include flaws that may negatively affect the reliability and resilience of these programs as individuals. A departure from the software requirements or specifications is a typical definition of an SD. Such flaws might result in failures or unexpected outcomes. The number of software failures may be reduced, and the program’s quality can be improved via a variety of software quality assurance activities. Techniques for SD predictions aid in locating software system components that are more probable to have flaws. Model assessing software modules by anticipated quantities of defects, defect likelihood, or classification outcomes may be created using DP methods (Qiao et al 2020). By examining the problem of SD, we suggested using ML and an E-ANN to forecast SD, which enhances testing and software quality. Particle swarm optimization and grey wolf optimization are used in grey wolf swarm optimization (GWSA) to maximize their complementary benefits and problems. The application of Ensemble Learning (EL) for the prediction of SD is the topic of five research issues covered in this article. After a comprehensive, methodical search procedure, the 46 articles that are the most pertinent to the issues at hand are selected. The research findings can be used as a standard for future enhancements and in-depth analyses while offering brief details about the most current trend and breakthroughs in EL for SDP (Matloob et al 2021).

Liang et al (2019) developed a Long Short-Term Memory (LSTM) network; users will need to use the vector sequences and their labels. The LSTM model can do fault prediction and effectively learn the semantic content of programs. According to the assessment findings on eight open-source projects, Seml beats three cutting-edge DP algorithms on most data for both within and cross-project defect predictions. Introduce a hybrid strategy by merging deep neural networks (DNN) for classification with genetic algorithms (GA) for characteristic optimization. A novel method for chromosomes constructing and computing fitness functions is included in an updated version of the GAA adaptive auto-encoder, another improvement made to the DNN approach that better represents certain software aspects. Case studies show that the suggested hybrid approach’s increased effectiveness results from optimization technology (Manjula and Florence 2019). Dam et al (2019) discussed the practical utility of a novel deep learning tree-based DP model. The Abstract Syntax Tree (AST), a depiction of the source code, is directly matched by the tree structured Long Short-Term Memory (LSTM) network on which the model mentioned above is constructed. As they create the model and test it on two datasets, one from open-source projects supplied by their client Samsung and the other from the public PROMISE repository, they share several things they have learned. The profession of software engineering is now primarily interested in the early detection of software problems. Several SDP strategies based on software metrics have been implemented over the last two decades. In predicting faults, bagging, decision trees (DS), and random forests (RF) classifiers are known to perform effectively. Alsaeedi and Khan (2019) analysed and contrasted these supervised ML and ensemble classifiers. The experimental findings demonstrated that, compared to the others, RF was the classifier that performed the best in most instances. Jayanthi and Florence (2019) presented a concept for feature reduction as well as artificial intelligence, with features reduction being carried out using the commonly used Principal Component Analysis (PCA) method and further enhanced by the addition of estimation of the maximal likelihood for the purpose of defect reduction in PCA data reconstruction. The DP, through the attention-based recurrent neural network (DP-ARNN) present in the present work, can forecast defects. To be more precise, DP-ARNN first extracts vectors from the program’s abstract syntax trees (ASTs). Then, using word embedding and dictionary mapping, it encodes the vectors that are utilized as the inputs for the DP-ARNN (Fan et al 2019). Balogun et al (2019) The repository of the National Aeronautics and Space Administration (NASA) was used in order to acquire the five SD datasets that were required for the assessment of the fourteen-filter feature subset selection (FSS) techniques and the four-filter feature ranking (FRR) methods. The effectiveness of the forecasting models used in the FFR approaches was noticeably boosted by the addition of Information Gain. The study provides a one-of-a-kind model by using a local tangent space alignment support vector machine (LTSA-SVM) approach, allowing it to handle the problem of software failure prediction. The SVM technique is used throughout the modelling process as the principal classification for the SD distributed prediction model. The model parameters are then improved by combining the grid search method with ten-fold cross-validation. This process is repeated 10 times. The accuracy of SVM is decreased in classic dimensionality reduction techniques due to data loss brought on by the poor characteristics of data nonlinearity (Wei et al 2019). Majd et al (2020) provided a novel method called Statement-Level SD prediction using the Deep-learning model (SLDeep). Because it shows a unique application of deep-learning models to resolve a real-world issue encountered by software engineers, SLDeep is significant for intelligent and expert systems.

2. Methodology

Stable SD predictive technologies and dynamically SDP technology are the two primary subcategories that make up SDP technology. The diverse aims and findings of the study are used as the basis for making these differences. Analyses are performed on the static SDP technology, with the goal of creating SD via the use of chronological data and metrics. The depiction is used to do an analysis of the software module’s tendency to have defects.
2.1. Static SDP Technology

The analysis of the software element convention, creation of the relevant dimension component, establishment of an opposite SDP structure constructed based on the measurement factor, and application of created SD are steps involved in the static SDP technology. Figure 1 shows that prediction results often come in two types: faults and non-defects. Software static measurement data and SD have a clear nonlinear connection that neither follows the established exact model nor is an essential combination of elementary nonlinear association methods.

![Figure 1 Basic rocket ship design.](https://www.malque.pub/ojs/index.php/msj)

2.2. Data Set for Training

There are two pieces to the software capacity and fault data. The static capacity data of each software function makes up one component, and the fault label (0 or 1) of each software function makes up the other. A web crawler tool was developed to gather the code that needs to be determined and defect details from open-source websites. Additionally, fixed metrics using standard concepts and Test Bed were expanded to compensate for the limited number of statistics and documents in the existing data set.

2.3. Basic Software Measurement

The static metrics have been concluded based on general principles, and Test Bed is an extensively used tool for doing fixed examinations on software. Table 1 has a complete listing of the metrics.

<table>
<thead>
<tr>
<th>Category</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC metrics</td>
<td>LOCphy, LOCComment, LOBlank, LOCComment</td>
</tr>
<tr>
<td>Dataflow information</td>
<td>Globals in procedure, File fan in, Fan out</td>
</tr>
<tr>
<td>Complexity metric</td>
<td>Knots, Quantity of loops, the intensity of loop nesting</td>
</tr>
<tr>
<td>Procedure information</td>
<td>process entry points, process exit points, entire comments</td>
</tr>
</tbody>
</table>

2.4. Cutting Software into Function

The syntax and semantic requirements for process-oriented languages are quite stringent. The languages must also adhere to formal programming and coding standards regarding the design of functions and the composition of code. These
grammatical and semantic requirements effectively improve the reliability of SDP by utilizing technology that provides code Function Level (FL) slicing and slicing criteria for code FL slicing.

2.5. Data collection and fusion for training

The method of data gathering based on web crawlers is shown in Figure 2 by using the process-oriented C language program as an example. The process shown in Figure 2 primarily consists of three elements. The first is to create function-level software measurement findings for the C language using code-slicing technologies. The second option is to use web crawler technology to automatically gather and recognize code and defect data gleaned from free software websites. The third phase entails the creation of measurements and defect data for a program written in the C language via the informational verification of function names. In order to offer data that can be used to produce superior and practical measures for SDe and defective details for SD prediction models, the data production process makes use of self-developed software tools. These metrics may then be employed in the models. These technologies feature high information excellence, rapid velocity, exact comparison, and function-level information granularity.

2.6. Enhanced Artificial Neural Network (E-ANN)

E-ANN, utilized in adaptive learning systems, may approximate a nonlinear input and output data transition. The outputted numbers indicate factors that are challenging to measure but are believed to be intimately dependent on the input variables. E-ANN is a powerful supervised learning model that has been used in a few areas, including the study of chemical interactions and the prognosis of SD. E-ANN adaptability enables independent parameter changes all through a training phase. In an SL scenario, pairs of specific instances are given to the network during training. An external supervisor provides the targeted output vector and the corresponding input vector. While the E-ANN model was built, it is learned and can offer an output vector based on an input instance that has not yet been seen (Figure 3).
A feed-forward E-ANN comprising many ultimately linked layers of neurons is called a multilayer perceptron (MLP). The backpropagation algorithm, an inductive learning method, trains the MLP. The fault discovered is sent back into the network, and its settings are modified during the input-output example given to it. Until a satisfactory performance is achieved, this procedure is repeated. The radial basis function network (RBFN) is a different neural network design. Mitchell claims that the output of the hidden units is dictated by a Gaussian activation function centred at a particular input instance. The output of an RBFN is the linear result of the hiding unit activations. The structure of the assumption that an RBFN learns provided an input $u$ is illustrated in equation (1).

$$
\hat{e}(u) = \sum_{l=1}^{L} \phi_l(c(d_{local}.u))
$$

In equation (1), $L$ stands for a specified number of Radial Basis (RB) functions, $d_l$ is the center selected for the $L_{th}$ RB function, and $\nu_l$ are positive real integers indicating the parameter values of the RB functions.

The metric $d$ is often the Euclidean distance, and the function $\phi_l(c(d_l..u))$ is frequently chosen as a Gaussian function focused on the position $d_l$ with a variance $\sigma_l^2$ (equations (2 and 3)).

$$
\phi_l(c(d_l..u)) = \frac{e^{-\frac{c^2(d_l.u)^2}{2\sigma_l^2}}}{\sigma_l^2}
$$

$$
\sigma_l = \alpha \frac{1}{m} \sum_{j=1}^{m} c(d_l..d_j)
$$

2.7. Grey Wolf Swarm Optimization Algorithm (GWSA)

The GWSA resolves several industry optimization issues and effectively produces very competitive outcomes. The GWSA algorithm is based on the SD. The GWSA splits its SD into four groups based on the prevailing hierarchical leadership order: alpha ($\alpha$), beta ($\beta$), delta ($\delta$), and omega ($\omega$). Consequently, the leaders with the highest ranks corresponding to the top three potential outcomes in the search field serve as the direction for prediction and optimization processes. These leaders are, respectively, $\alpha$, $\beta$, and $\delta$.

The remainder of the solutions is represented by the $\omega$ SD, which is the smallest in the hierarchy and must change its position to accommodate the other dominating DP. Provided that it is expected that each alternative solution of dimensions $n$ be expressed by the vector $\vec{U}$, the location vector of the grey wolf is provided in equation 4 as follows:

$$
\vec{U} = \{u_1, u_2, ..., u_m\}
$$

The GWSO and DP model combination can improve defect management tactics and reduce the effects of software problems, thereby boosting the quality and dependability of software systems. Equations 5 and 6 could be used to depict this behavior in GWSA quantitatively:

$$
\vec{U} = |\vec{B} \cdot \vec{U}(s) - \vec{U}(s)|
$$

$$
\vec{U}(s + 1) = \vec{U}_o(s) + \vec{B} \cdot \vec{C}
$$

where $\vec{U}_o$ is the position vector of the defects, $\vec{U}(s)$ is the position vector of the Grey wolf, $s$ is the current iteration, $\vec{B}$ and $\vec{E}$ are coefficient vectors vary to allocate the wolf to adjust their positions in the space approximately the defects. The coefficient vectors $\vec{B}$ and $\vec{E}$ are computed according to equation 7.
\[
\vec{b} = 2\vec{b}\vec{q}_1 - \vec{b} \\
\vec{e} = \vec{q}_2
\]  

(7)

The elements \(\vec{b}\) are assumed to linearly decrease within an initial value of 2 to a value of 0 during the search process, while \(\vec{q}_1\) and \(\vec{q}_2\) are random vectors chosen from the range [0,1]. Then, the GWSA preserves the top three solutions and lets the remaining solutions move into place following the best solutions’ locations. To get the difference between the present location and \(\alpha\), \(\beta\), and \(\delta\), respectively, equation 8 is employed.

\[
\begin{align*}
\vec{c}_\alpha &= |\vec{d}_1, \vec{u}_\alpha - \vec{u}| \\
\vec{c}_\beta &= |\vec{d}_2, \vec{u}_\beta - \vec{u}| \\
\vec{c}_\delta &= |\vec{d}_3, \vec{u}_\delta - \vec{u}|
\end{align*}
\]

(8)

where \(\vec{u}\) is the location of the current solution, \(\vec{d}_1\), \(\vec{d}_2\), and \(\vec{d}_3\) are random vectors, and \(\vec{u}_\alpha\), \(\vec{u}_\beta\), and \(\vec{u}_\delta\) are the locations of the \(\alpha\), \(\beta\), and \(\delta\) respectively. Then, using equation 9, it is possible to determine the final location of the present solution.

\[
\begin{align*}
\vec{u}_1 &= |\vec{u}_\alpha, \vec{d}_1 - \vec{c}_\alpha| \\
\vec{u}_2 &= |\vec{u}_\beta, \vec{d}_1 - \vec{c}_\beta| \\
\vec{u}_3 &= |\vec{u}_\delta, \vec{d}_3 - \vec{c}_\delta|
\end{align*}
\]

(9)

Thus, \(\vec{u}(s + 1)\) can be computed as follows (equation 10):

\[
\vec{u}(s + 1) = \frac{\vec{u}_1 + \vec{u}_2 + \vec{u}_3}{3}
\]

(10)

where \(s\) represent the number of iterations, and \(\vec{B}_1\), \(\vec{B}_2\), and \(\vec{B}_3\) are random vectors that vary to allow the defects. SD may significantly impact software system performance and dependability. Researchers have recently been experimenting with different optimization methods to improve error identification and prevention. Using the GWSA algorithm is one option that shows promise. GWSA, which derives its cues from the social interactions of unclear, may be used to improve the settings and characteristics of DP models like ANNs. The performance of DP models may be enhanced by GWSA, resulting in more precise and trustworthy detection of SD.

3. Result and discussion

Software quality and testing may be enhanced by using ML and an Enhanced Artificial Neural Network (E-ANN) to anticipate SD. The grey wolf swarm optimization algorithm (GWSA) combines particle swarm optimization with grey wolf optimization to maximize their complementary benefits and downsides. A suggested method’s effectiveness is evaluated with that of existing techniques such as K-Nearest Neighbor’s algorithm (KNN) (Mabayoje et al 2019), Support Vector Machine (SVM) (Goyal 2022), and Convolutional Neural Network (CNN) (Pan et al 2019). These techniques are compared with previous techniques using several parameters, including accuracy, precision, recall, and f1 score.

3.1. Accuracy

One of the most often used measures for assessing classifier models is accuracy, which reflects the total impact of prediction. Still, using accuracy as the criterion for measuring a prediction model’s effectiveness in DP is pointless because of the severe division inequity problem in the defect data.

\[
\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}
\]

(11)

Figure 4 and Table 2 shows the accuracy of the proposed and existing system. EANN can be used as a predictive model to detect or categorize SD, and GWSO can be used to optimize the parameters or features of the EANN to increase its accuracy. KNN has attained 0.76%, CNN has attained 0.88%, SVM has acquired 0.82%, and the proposed system reached 0.94% accuracy. It demonstrates that the suggested method is more accurate than the existing one.

<table>
<thead>
<tr>
<th>Table 2 Accuracy.</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN (Pan et al 2019)</td>
<td>0.88</td>
</tr>
<tr>
<td>KNN (Mabayoje et al 2010)</td>
<td>0.76</td>
</tr>
<tr>
<td>SVM (Goyal et al 2022)</td>
<td>0.82</td>
</tr>
<tr>
<td>E-ANN+GWSA [Proposed]</td>
<td>0.94</td>
</tr>
</tbody>
</table>
3.2. Precision

Precision in the context of SD refers to the percentage of defects accurately recognized out of every reported defect. It highlights the capacity to prevent false positives by evaluating the reliability and accuracy of fault detection.

\[
\text{Precision} = \frac{TP}{TP + FP} \quad (12)
\]

Figure 5 and Table 3 shows the precision of the proposed and existing system. GWSO is an optimization method to improve the EANN's features or parameters. Although it has little immediate effect on precision, it might make the EANN function greater general. KNN has attained 0.69%, CNN has attained 0.74%, and SVM has attained 0.83%, whereas the proposed system reached 0.92% accuracy. It demonstrates that the suggested method is more precise than the existing one.

3.3. Recall

Recall, as referring to SD, is the percentage of defects accurately recognized out of all the actual defects present in the software system. It gauges whether comprehensive and accurate fault detection is, demonstrating the capacity to prevent false negatives.

\[
\text{Recall} = \frac{TP}{TP + FN} \quad (13)
\]

Figure 6 and Table 4 shows the recall of the proposed and existing system. It concentrates on the EANN's capability to accurately discover or identify genuine faults from the available data while taking into account the recall of an EANN with
the aid of GWSO for SD.KNN has attained 0.79%, CNN has attained 0.85%, and SVM has attained 0.82%, whereas the proposed system reached 0.96% accuracy. It demonstrates that the suggested method is more recall than the existing one.

<table>
<thead>
<tr>
<th>Table 4 Recall.</th>
<th>Recall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN (Pan et al 2019)</td>
<td>0.85</td>
</tr>
<tr>
<td>KNN (Mabayoje et al 2010)</td>
<td>0.79</td>
</tr>
<tr>
<td>SVM (Goyal et al 2022)</td>
<td>0.82</td>
</tr>
<tr>
<td>E-ANN+GWSA [Proposed]</td>
<td>0.96</td>
</tr>
</tbody>
</table>

3.4. F-measure

A statistic often used to assess the overall effectiveness of fault detection or classification systems is the F-measure, sometimes known as the F1 score. It provides a balanced measurement of both measures by combining recall and accuracy into a single number.

$$F1 - score = 2 \cdot \frac{Precision \times Recall}{Precision + Recall}$$ (14)

Figure 7 and Table 5 shows the F-measure of the proposed and existing system. Network design, enhancement methods, and optimization parameters affect EANN with GWSO. To estimate the F-measure and performance of the EANN with GWSO for SD prediction, experimentation, and assessment are necessary. KNN has attained 0.78%, CNN has attained 0.71%, and SVM has attained 0.85%, whereas the proposed system reached 0.91% accuracy. It demonstrates that the suggested method is more F-measure than the existing one.

<table>
<thead>
<tr>
<th>Table 5 F-measure.</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN (Pan et al 2019)</td>
<td>0.71</td>
</tr>
<tr>
<td>KNN (Mabayoje et al 2010)</td>
<td>0.78</td>
</tr>
<tr>
<td>SVM (Goyal et al 2022)</td>
<td>0.85</td>
</tr>
<tr>
<td>E-ANN+GWSA [Proposed]</td>
<td>0.91</td>
</tr>
</tbody>
</table>
4. Conclusions

Software defects, additionally referred to as software bugs or software faults, are defects or errors in the code of a piece of software that might cause it to behave in a manner that is not consistent with what it was intended for it to do or to create outcomes that are erroneous or unanticipated. Errors in logic or syntax, inconsistencies in the design, or problems with compatibility are some of the ways that these faults may appear. The use of ML and an Enhanced Artificial Neural Network (E-ANN) to predict SD is one way to improve software quality and testing. The grey wolf swarm optimization method (GWSA) combines particle swarm optimization with grey wolf optimization to optimize their complementing advantages and drawbacks. SD constitutes a substantial barrier in SDe owing to many fundamental restrictions. Due to time and resource limitations, limited testing coverage makes the issue worse by making it impossible to test software in all scenarios and configurations. SDe groups should continue to place a high priority on fixing SD in their future work. It is crucial to keep researching cutting-edge methods to identify, avoid, and manage software faults, given the complexity of software systems and the rising need for dependable and secure applications.

Ethical considerations

Not applicable.

Declaration of interest

The authors declare no conflicts of interest.

Funding

This research did not receive any financial support.

References


