Journal of Theoretical and Applied Information Technology

15th October 2023. Vol.101. No 19

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A HYBRID GENETIC BASED GREY WOLF OPTIMIZED SOPHISTICATED SUPPORT VECTOR MACHINE (SSVM) MODEL FOR SOFTWARE DEFECT PREDICTION

DR MEDHUNHASHINI[1], Dr. KS JEEN MARSELINE[2]

Research Scholar, Sri Krishna Arts and Science College,

Dean & Head, Sri Krishna Arts and Science College,

E-Mail : [1]medhun.hashini@gmail.com, [2]jeenmarselineks@skasc.ac.in

ABSTRACT

Software Defect Prediction is one of the promising fields in software engineering, focusing on identifying and predicting the defective module in software before the testing phase begins. It helps to allocate resources in the testing phase cost-effectively. Developing a machine learning model that classifies the faulty module from non-faulty seems challenging. This paper focuses on developing an ensemble machine learning model, a Sophisticated Support Vector Machine (SSVM), for effective defect prediction. SSVM is built with the hybrid power of GA and GWO over the SVM. An enhanced Genetic Algorithm (GA) is used to select appropriate features from the defect dataset by Crossover of selected features. Grey Wolf Optimization (GWO) has been adopted to tune the hyperparameter of SVM's Radial Basis Function Kernel. The defect dataset JDT and MyLyn from the AEEEM repository is taken for experimentation. The model is investigated with 10-fold cross-validation, and performance is evaluated with a confusion matrix and F1 score. The results show the SSVM model classifies the defective from the non-defective module with an accuracy of 75.30 % and 77.30 %.

Keywords: *Quality, Software Defect, Genetic Algorithm, Grey Wolf Optimization, Support Vector Machine*

1. INTRODUCTION

Software Defect Prediction (SDP) plays a crucial role in software development by enabling development teams to make informed decisions and take proactive measures to improve software quality. By identifying potential problem areas early in development, teams can allocate resources effectively and prioritize testing efforts. This helps them focus on compulsory modules or components more likely to contain defects, reducing the risk of releasing software with significant issues[1]. One of the critical advantages of SDP is the ability to allocate resources effectively. Development teams often have limited resources, including time, budget, and human resources. By leveraging defect prediction models, teams can identify high-risk areas and allocate their limited resources accordingly. This ensures that the most critical modules or components receive greater attention regarding code reviews, testing, and debugging. By doing so, teams can address potential issues before they escalate and cause more significant problems[2].

SDP allows teams to prioritize their testing efforts. Testing is critical to software development but can be resource-intensive and time-consuming. By predicting potential defects, teams can prioritize testing efforts on modules or components more likely to contain issues. This targeted approach helps ensure that critical functionality is thoroughly tested, reducing the risk of defects slipping through the testing phase and being discovered later by the end-user [3]. SDP helps improve software quality by identifying potential problems before they manifest as defects. By analyzing historical data and applying machine learning algorithms, these models can uncover patterns and relationships between various factors and the occurrence of defects. This insight enables development teams to identify familiar sources of defects and implement preventive measures, such as code refactoring, process improvements, or additional quality assurance activities [4].

By addressing potential issues early on, teams can significantly reduce the likelihood of defects and improve the overall quality of the software. It is important to note that SDP models are not foolproof. They rely on historical data, assumptions, and correlations, which may not capture all potential sources of defects. As software systems and development practices evolve, these models require regular updates and refinements to

remain effective. New defects may emerge, and the models must adapt to these changes to provide accurate predictions. The success of defect prediction also depends on the quality and relevance of the historical data used for training the models. Therefore, the models' ongoing monitoring, evaluation, and improvement are necessary to ensure their effectiveness over time[5].

SDP empowers development teams to allocate resources effectively, prioritize testing efforts, and improve software quality. By identifying potential problem areas early in the development process, teams can take proactive measures to mitigate risks and address issues before they impact end users [6]. While not infallible, defect prediction models provide valuable insights that help teams make informed decisions and continuously enhance software quality.

1.1. Machine Learning

Machine Learning (ML) plays a significant role in SDP, revolutionizing how software systems identify and address defects. By leveraging advanced algorithms and statistical models, ML techniques can analyze large volumes of data, detect patterns, and make accurate predictions. In the context of SDP, ML enables the development of sophisticated models that can effectively forecast potential defects and guide developers in taking proactive measures. One key aspect where ML excels is analyzing historical data. By feeding large sets of historical defect data into ML algorithms, models can be trained to learn from past experiences and uncover hidden patterns and relationships. These patterns can reveal valuable insights into the factors contributing to defects, such as code complexity, code churn, coding standards violations, and historical bug fixes[7]. By recognizing these patterns, ML models can identify areas of the codebase that are more likely to contain defects, enabling developers to allocate their resources effectively and focus their efforts on those specific areas. ML algorithms also handle complex and non-linear relationships between software metrics and defects. Traditional statistical methods may struggle to capture the intricate interactions between variables, but ML techniques can effectively capture these complex relationships. For example, decision trees, random forests, support vector machines, and neural networks are commonly used ML algorithms in defect prediction[8]. These algorithms can simultaneously consider multiple software metrics and identify non-obvious combinations of factors contributing to defects, leading to more accurate predictions.

ML models can adapt and improve over time. As new data becomes available, models can be retrained to incorporate the latest information, allowing them to evolve and enhance their predictive capabilities continuously. This adaptability is crucial in the dynamic field of software development, where new types of defects may emerge, and software systems may change. ML models can be updated to reflect these changes and maintain their effectiveness in predicting defects. ML techniques also enable the integration of various data sources for defect prediction. In addition to code-related metrics, ML models can incorporate data from other sources, such as bug-tracking systems, version control systems, and developer collaboration platforms [9]. By incorporating multiple data streams, models can comprehensively understand the software development process and capture a broader range of factors contributing to defects. It is important to note that ML models for defect prediction require careful consideration and validation. The quality and relevance of the training data, feature selection, and appropriate model evaluation techniques are crucial to ensure the accuracy and effectiveness of the models. Additionally, the interpretability of ML models is an ongoing research challenge [10]. While models may achieve high predictive accuracy, understanding the underlying reasons for their predictions can be challenging. Interpretable ML methods are actively being explored to address this issue.

1.2. Problem Statement

The challenge of dataset representativeness in SDP arises when the dataset used for training and testing fails to accurately capture the diverse characteristics and variations of the target software system. This can lead to unreliable predictions, false positives or negatives, and hinder the overall quality and reliability of the software. Inadequate sampling, biases in data collection, and overlooking variations in defect patterns across different software components or development stages contribute to the lack of dataset representativeness. Addressing this challenge requires careful selection and collection of data that accurately reflects the system's diversity, ensuring the development of reliable defect prediction models that support adequate software quality assurance.

1.3. Motivation

The motivation behind addressing the challenge of dataset representativeness in SDP lies in the quest for accurate defect predictions, improved software quality, and enhanced software engineering practices. By ensuring that the dataset

used for training and testing prediction models accurately represents the target software system, we can develop robust models that capture variations in defect occurrence, enabling proactive defect identification and mitigation. This, in turn, leads to enhanced software reliability, reduced maintenance costs, and increased end-user satisfaction. Additionally, addressing dataset representativeness fosters advancements in research and practice, enabling the development of more reliable prediction algorithms and supporting informed decision-making for resource allocation and risk management. Ultimately, the goal is to elevate software quality and reinforce the development process to deliver reliable and robust software systems.

1.4. Objective

The objective of this research is to propose an ML algorithm for SDP that addresses the challenge of dataset representativeness. Developing an algorithm that can effectively leverage representative datasets aims to improve the accuracy and reliability of defect predictions in software systems. The specific objectives include:

- ✓ identifying and implementing strategies to ensure dataset representativeness, including appropriate sampling techniques and bias mitigation approaches;
- ✓ designing and implementing an ML algorithm that effectively utilizes the representative dataset for defect prediction;
- ✓ evaluating the performance of the proposed algorithm against existing approaches using comprehensive metrics and real-world software datasets;
- ✓ analyzing the impact of dataset representativeness on the algorithm's predictive capabilities; and
- ✓ demonstrating the practical utility of the proposed algorithm by highlighting its potential for proactive defect identification, enhanced software quality assurance, and improved decision-making in software development and maintenance processes.

The significant objective is to contribute to advancing SDP techniques by addressing the critical challenge of dataset representativeness.

1.5 Organization of the Work

This research paper is organized as in Section 2, a description involving the various literature on software defect prediction, their adapted algorithms and the performance measures that are prominently used in this domain. Section 3 elaborates on the proposed SSVM defect prediction model in which the SVM is used as a classifier, including an enhanced genetic algorithm for feature selection and GWO for hyperparameter tuning the radial basis function kernel of SVM. Section 4 gives an insight into the dataset's origin, its features and its representativeness. Section 5 discusses the results of the experiments conducted using SVM, SVM-GA, and SSVM models.

2. LITERATURE REVIEW

The "Effort-Aware based Defect Prediction model"[11] is constructed by EALTR using the linear regression model, which is then used to develop a set of coefficient vectors for the linear regression model using the composite differential evolution algorithm. To build the EADP model, EALTR chooses the coefficient vector with the highest PofB@20% value on the training dataset. They suggest a re-ranking technique in the prediction phase to further minimize the Initial False Alarms (IFA) value of EALTR. "Deep learning-based prediction" [12]used qualitative and quantitative analysis of the data extracted for Defect Prediction. It concluded that most of the defect predictions employed Supervised learning using the software metrics as data feed amongst the CNN, which is mostly used deep learning technique for Defect Prediction. "Ensemble Machine Learning" [13], a review was conducted on the hybrid defect prediction models and concluded that an ensembled machine learning technique can be used to develop a robust hyperparameter optimization for better defect prediction.

A "Transfer-learning Technique (TLT) based defect prediction" [14]is used to find the code clone in the cross-functional projects to reduce the clone-consistent defects. TLT shows that the quantity of the dataset also has a favourable impact on prediction and that transfer-learning approaches have a positive impact on forecasting cross-project clones consistent-defect in the early prediction of defects in the software development life cycle. The "Hybridized Machine Learning Algorithm"[15] model for software failure prediction was involved in selecting characteristics with a better fitness function using a genetic algorithm (GA) to optimize the data set features. A Decision Tree supervised learning strategy is used as a classification method to analyze the features after choosing the best ones. The available machine learning models are compared for the performance with GA-DT based model to predict error proposed as RCSOLDA-RIR and WPA-PSO models. The experimental analysis's

findings show that the proposed model performs better in terms of accuracy than the one currently in use.

"Issues-Driven Feature for Software Defect Prediction" [16] was experimented on 86 open-source projects from two organizations and then experimentally compared the performance with cutting-edge traditional features. The findings demonstrate that Issues-Driven features outperform state-of-the-art features and result in an AUC improvement of 6 to 13%. Their study concluded that including the needs into fault, prediction features circumvents the drawbacks of conventional software metrics that are indifferent to the software's requirements. "ARRAY: Adaptive triple feature-weighted transfer Naive Bayes for cross-project defect prediction" [17] consisted of Feature weighted similarity, feature-weighted instance weight, and model adaptive adjustment as its characteristics. They used 34 faulty datasets for their experiments. Comparison of ARRAY with other Cross Project Defect Prediction (CPDP) methods was made using several statistical techniques such as ROC curve (AUC), F1, and Matthews Correlation Coefficient (MCC). Their results open up ARRAY to have a significant improvement in MMC, AUC and F1 values by 18.4%, 6.5%, and 4.5% and perform better than any base value.

"Machine Learning-Based ensemble methods" [18]for data pre-processing, feature selection further algorithm implementation to predict the software defect and improve software quality was used. Software metrics from three datasets were used for functioning. The end results showed that logistic regression on the EBSPM Dataset gave 96.67% accurate prediction, and other methods like Soft Voting and Stacking (SVS), Gradient Boost and bagging produced the best results of 96.31% and 94.59% on the PROMISE Repository Software metric data. "Data sampling and feature selection techniques for software fault prediction" considered eight sampling techniques involving ten feature selection algorithms in the open-source projects were performed, and the accuracy was estimated. ROC and AUC metrics were used to find the performance. According to the empirical findings, the Synthetic Minority Over Sampling Technique Edited (SMOTEE) and correlation-based feature selection (FS2) combination outperformed the highest AUC value for the projects. The experiments showed a 24.07% of projects that used the SMOTEE, FS2, and RF combination were able to achieve the greatest AUC values.

"Imbalance Class Learning using Weighted Average Centroid" [19] WACIL for diversity imbalance learning is used for an efficient synthetic oversampling method to address the defect imbalance problem. The WACIL first identifies baseline cases, then produces false data of them using a weighted average centroid concept and performs a filtration process to remove unsuitable noise data. Comparing WACIL to competing other prediction algorithms, its performance is achieved by finding the Fall Out Rate (FOR), F-measure, and Area Under Curve (AUC), while achieving equivalent results in terms of Recall and G-mean. The "African buffalo optimized model for software fault prediction" [20] approach aims to increase software dependability helping to find the fundamental reason for software failure at an early stage. The ABOMSR-CDNL Model has four levels: an input layer, two hidden layers, an output layer, and software programme codes. A software's log files serve as an input layer for the model's first hidden layer, which receives the software programme codes. A project portfolio is created using the best parameters found in the software event log files and then sent to the hidden layer 2. This technique examines system application failure behaviours in a shorter length of time by the application of multinomial softmax regression analysis by the ABOMSR-CDNL Model.

"A fuzzy logic expert system to predict module fault proneness using unlabeled data rule-base and database" [21]overcame the Takagi and Sugeno-based fuzzy logic system that was manually created initially and later enhanced by prioritizing the module's flaws. The second method can determine the ideal threshold values and provide the best recommendations for how to conduct testing activities in order to increase the calibre of software testing on a limited budget and schedule. "Hybrid Training Data Selection (HTDS)"[22] method that combines feature selection and instance selection to create a competent and practical CPFP mode is used to validate this method's efficacy using NEPMs and effort-based performance measures (EPMs) to ensure that it is applicable in real-world settings. The ethnicity of Cross Project Defect Prediction and the validation of the model for quality improvement based on NEPM becomes tedious with unlimited resources.

3.SOPHISTICATED SUPPORT VECTOR MACHINE (SSVM)

Software defect prediction is highly dependent on software metrics, and reliable metrics

envision great predictability. The proposed work encompasses a hybrid Genetic based Grey Wolf Optimized Sophisticated Support Vector Machine (SSVM) hyperparameter selection model for predicting defects in software.

3.1 Support Vector Machine

Support Vector Machine is a promising machine learning algorithm for defect prediction, preferably used for issues in binary classification. A kernel function of the SVM separates a hyperplane from the margin value. Three kernel functions can be used, and each is significantly different from the others. A linear kernel binary classification of the data points in the search space is placed with the help of Eq (1)

$$f(x) \rightarrow x' \quad (1)$$

An alternative approach to fix the data points is to use a polynomial kernel function, as stated in Eq (2). It margins aptly to draw the hyperplane in the search space. The distance between the data points and the hyperplane is quite lesser and thus covers most of the points into the space where d is the degree and r is the coefficient.

$$p_k = (\gamma(x, x') + r)^d \quad (2)$$

A non-linear connection between the data points can be handled effectively using a polynomial kernel. High-dimensional data points with lesser samples are efficiently handled. Amongst the linear and polynomial kernel functions that exist, RBF maximizes tuning the hyperparameter functionality and is effectively used in SVM for defect prediction comparably. Instead of using the empirical method for training data, the kernel function produces a set of training data independently. The first step to implement is to choose the kernel function and map to the feature of a dataset to a high dimension space **H**. It is mathematically expressed as Eq (3).

$$\begin{aligned} &\emptyset: x \rightarrow H \\ &x \rightarrow \emptyset(x) \end{aligned} \quad (3)$$

The classification of the dataset is dependent on hyperparameter C. Overfitting the C parameter results in unbiased and varied results. Fitting the value of C will increase the prediction accuracy. The RBF kernel function of SVM is implemented as in Eq (4).

$$k(p, q) = \exp(-\frac{||p - q||2}{2\sigma^2}) \quad (4)$$

From Eq.(2), σ is the hyperparameter and $||p - q||2$ is the Euclidean distance calculated between the input points p and q. Setting the hyperparameter C and γ optimally to find the solution is usually a problem. To solve this, tuning the parameter is required. A genetic algorithm combined with Grey Wolf Optimization becomes a powerful mechanism proposed to find the optimal parameter value for the hyperparameters of SVM.

3.2 Genetic Algorithm

An SVM classifier's gamma parameter γ can be adjusted with a genetic approach. The gamma parameter controls how much each training sample has an impact on the SVM's decision border. The gamma value maximizes the SVM classifier's performance by employing a genetic algorithm.

The genetic algorithm (GA) is an adaptive search algorithm for the optimization issue. Here genetic algorithm proposes a hyperparameter set generated randomly and is used as the empirical model. Each generated set solution is investigated for fitness, knowing its accuracy level. Among the generated solution set, the best population is found by a pruning procedure that uses Grey Wolf Optimization (GWO).

This metaheuristic algorithm was influenced by the ideas of genetics and natural evolution. Problems with search and optimization are frequently solved with it. GA identifies an ideal or nearly ideal solution by iteratively evolving a population of candidate solutions. The working of GA is provided in Algorithm 1.

A ***random population*** set of solutions are taken as and possibly fits into the search space r specified as in Eq (5)

$$P_{set} = \{p_1, p_2, p_3 \ldots . p_n\} \quad (5)$$

A ***fitness function*** of each chromosome is the quality of the chromosome that is the solution's correctness that competes with others to prove them best and is defined as in Eq (6) where r is the search space

$$f = r \rightarrow R \quad (6)$$

Based on the fitness value of each individual, find the selection probability. Consider $SP(p)$ as the selected probability of individual p. The GA is outlined in Algorithm 1.

Algorithm 1: GA

Step 1: Generate a random population set
Step 2: Initialize ***population(P)***
Step 3: Evaluate the fitness of each individual in the population
Step 4: While termination condition not met:
- Select parents from a ***population(P)***
- Perform Crossover to create ***offspring (CO)***
- Perform Mutation on the ***offspring (MO)***
- Select individuals for the next generation based on fitness

Step 5: Return the best solution

One-point Crossover technique is used to randomly choose a point along the length of the chromosome to transfer between the parents to produce kids. The crossover equation is given in Eq (7). A new offspring is generated from the two randomly selected parents placing a point where the genetic material exchanges.

$$Offspring_1 = \sum_{i=0}^{2} Parent_i[CrossoverPoint] \quad (7)$$

A ***mutation*** process happens among the individuals leveraging random change in the population. A diverse population is then evolved to explore new search space. The Mutation is defined as in Eq (8).

$$X_{i=}Y_i + ||m|| * R \quad (8)$$

Where Xi is the new mutated individual set, Yi represents the chromosomes before Mutation, $||m||$ is the parameter that controls the mutation change, and R is the randomly generated range of values for Mutation.

The fitness value of each population is understood to know the next generation's evolution. The GA adaptatively optimizes the solution set until the termination criterion is met.

3.3 Grey Wolf Optimization

Grey Wolf Optimization *(GWO)* is a nature-inspired algorithm obtained from the hunting behaviour of Grey Wolves. Grey wolves show community behaviour living and hunting as a pack of 5-12 wolves together. They organize themselves in a hierarchy. The $Alpha\ \alpha$ is dominant among and is responsible for hunting the Prey and dictates the remain. The $Beta\ \beta$ wolf is the second dominant in the pack and is responsible for helping the alpha wolf make decisions.

The third dominant in the pack is $Delta\ \delta$ wolf. They have to respond to the decisions taken by the first two dominants. The remaining in the pack, $Omega\ \omega$, plays a subordinate role and is the least important in the pack. The GW is mathematically modelled as the fittest solution to any problem as an α wolf and the second-best solution as the β wolf, the third-best solution as δ wolf finally, the rest candidate solutions omega wolves.

The grey wolves exhibit a social hierarchy, and their working phase involves

- Encasing the Prey
- Surrounding and attacking the Prey to mortal

The GWO is outlined in Algorithm 2.

Algorithm 2: GWO

Step 1: Initialize the Grey Wolf population $\boldsymbol{X_i, where\ i = 1, 2, 3, \ldots . n)}$
Step 2: Initialize $\boldsymbol{\alpha, A}$ & $\boldsymbol{C}$
Step 3: Evaluate the fitness of individual members of the $\boldsymbol{Xi}$
Step 4: Assign
- $\boldsymbol{X_\alpha}$with the first best-fit value
- $\boldsymbol{X_\beta}$with the second fit value
- $\boldsymbol{X_\delta}$with the third fit value

Step 5: To $\boldsymbol{begin}$ to $max\_iterations$
Step 6: Update the position of $\boldsymbol{X_\omega}$ as in Eq 13,14,15
Step 7: Calculate the fitness of all search entity
Step 8: Update the positions of $\boldsymbol{X_\alpha, X_\beta, X_\delta}$
Step 9: Return the $\boldsymbol{X_\alpha}$

In the GWO algorithm, hunting takes place in phases that are evolved mathematically, as in Eq (7) and Eq (8).

3.3.1 Encasing the Prey

The primary and foremost phase before hunting the Prey is covering them for an attack. The α wolf initiates the process and commands the β and δ wolves. In the below Eq (9) and Eq (10)

$$\vec{D} = |\vec{C}.\vec{X}_i(t) - \vec{X}_i(t)| \quad (9)$$

$$\vec{X}(t+1) = \vec{X}_i(t) - \vec{A}.\vec{D} \quad (10)$$

Here $\vec{A}$ and $\vec{C}$ are the vector coefficient, t denotes the present iteration, the vector position of the Prey is denoted as $\vec{X}_1$, and $\vec{X}$ is the position of the Grey Wolf.

The two vectors $\vec{A}$ and $\vec{C}$ are calculated as in Eq(11) and Eq (12)

$$\vec{A} = 2\vec{a}.\vec{r}_1 - \vec{a} \quad (11)$$

$$\vec{C} = 2.\vec{r}_2 \quad (12)$$

where $\vec{a}$ value decreases from 2 to 0 during the iterations, and $\vec{r}_1$ and $\vec{r}_2$ are vectors with random values between 0 and 1.

3.3.2 Surrounding and Hunting the Prey

The α,β and δ wolves are positioned in the first three places for hunting the Prey, and the ω wolves are to be positioned as in Eq.(13) and Eq.(14).

$$\vec{D}_\alpha = |\vec{C}_1.\vec{X}_\alpha - \vec{X}|,\ \vec{D}_\beta = |\vec{C}_2.\vec{X}_\beta - \vec{X}|,\ \vec{D}_\delta = |\vec{C}_3.\vec{X}_\delta - \vec{X}| \quad (13)$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1.(\vec{D}_\alpha),\ \vec{X}_2 = \vec{X}_\beta - \vec{A}_2.(\vec{D}_\beta),\ \vec{X}_3 = \vec{X}_\delta - \vec{A}_3.(\vec{D}_\delta) \quad (14)$$

$$\vec{X}(t+1) = \coprod_{n=0}^{3}\left(\frac{\vec{X}_n + \vec{X}_{n+1} + \vec{X}_{n+2}}{3}\right) \quad (15)$$

In the Eq (11). $\vec{A}$ is a random value between the interval value $-2a$ and $2a$. When $|A| < 1$ results in the exploitation allowing the wolves to attack the Prey, and $|A| > 1$ tends to move from the current Prey to a new fittest prey.

GWO is a metaheuristics algorithm that efficiently optimizes the result for software defect prediction. The candidate solutions can still be prioritized so that when the α, β, and δ go unfit, the next one can take up the α position can be decided. The grey wolf optimization mimics the characteristics of the group and thus is highly dependent on each other. The candidate solutions can be further sequenced and prioritized by another wolf φ.

3.4 Hyperparameter tuning using Hybrid Genetic Algorithm based Grey Wolf optimization for SSVM

The main prerequisite for using SVM is that the kernel hyperparameters must be set properly in order for SVM to function. The C and γ kernel parameters are present. The functioning capacity of SVM is struck by a tradeoff parameter C to increase the forecast accuracy and stability. Classification accuracy using SVM significantly depends on the Radial Basis Function Kernel Gamma parameter γ.

The architectural flow of the SSVM model is shown in Figure 1. This model collaborates the power of GA and GWO for the feature selection and hyperparameter tuning of the RBF Kernel in SVM. The model starts working with the software defect data as the input. The SVM classifier prominently classifies the faulty and non-faulty modules from the data. Before the classification begins, the raw data is cleaned. Data preprocessing involves removing the null values and handling the missing values, as they may produce inappropriate results when employed. Outliers detected are removed.

Data is transformed into a form that is suitable for building an SSVM Model. Any categorical data seems to be unfit for the prediction model; hence conversion is employed. It is important to find the feature from the given data that are highly beneficial. The efficiency of the defect prediction model enhances with the finest features. SSVM model needs the γ and C parameters to be experimented with for the finest fit. The c parameter is the control regularization entity that fits the points in the search space at a marginal level. A higher value of C misclassification is lesser and helps obtain less margin. Lowering the value of C leads to the misclassifications of the data points with a higher margin. SSVM model has experimented with a range of values for C.

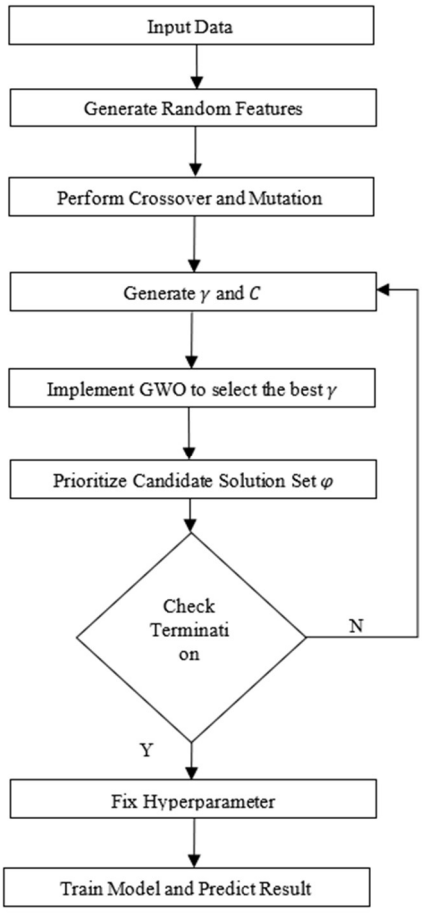

Figure 1 : Architectural Flow of SSVM Model for Defect Prediction

The γ specifies the decision boundary for the classification, and the value can be set between 0.1 to 1. The highest value represents the data points closer to the decision boundary, which makes a smooth curve; conversely, for the lowest value, the data points are away from the decision boundary. The selection of the γ parameter is optimized using GA. The GA algorithm involves Crossover and Mutation for placing the optimized points in the search space.

The GWO tunes the generated population set from the vector space. The fitness function of GWO is tuned for several iterations over the wolf population. Sort the fitness function of the wolves in ASC or DSC order. Fix the optimal population from GA for the best three wolves' alpha, beta and delta after sorting. The GWO optimizes the best population set achieved through GA, and further, the

hyperparameters of Support Vector Machine C and γ are fined tuned to obtain the best solution for Software Defect Prediction (SDP).

The SSVM model leverages the potential data points placing them in the search space. A hyperplane is placed with the boundary that provides the best results in the software defect prediction.

The $SSVM$ is outlined in Algorithm 3.

Algorithm 3: SSVM

Step 1: Load Dataset
Step 2: Implement enhanced GA for feature selection
Step 3: Select feature population set
Step 4: Perform Crossover and mutate
Step 5: Point the data in high dimensional search space
Step 6: Initialize the hyperplane value
Step 7: Set γ and C as in Eq (2)
Step 8: Optimize the γ with fitness function as in Eq (15)
- Implement the α value to γ
- Update β,δ,ω

Step 9: Prioritize the candidate solution and set φ
Step 10: Repeat Step 9 when α goes unfit
Step 11: Train the SSVM Model and Predict the result

4. ABOUT DATASET

The "MyLyn" and "JDT" dataset about software defect prediction is taken from the AEEEM (Appraisal-Based Estimation of Effort) repository. A dataset created especially for testing evolutionary algorithms used in software defect prediction. The datasets in the AEEEM repository were gathered from several open-source Java software projects. AEEEM is a benchmark dataset frequently used in software engineering research. The dataset contains software artefacts and their defect associated with each module.

A manual inspection is made, and also data collected from bug-tracking systems is investigated. The dataset is labelled to show whether each artefact contains a fault or not. The artefacts are converted into software metrics. Researchers use these datasets to evaluate the forecasting model created for defect prediction. Table 1 describes the datasets from AEEEM used for this study. Each of the datasets consists of 61 features associated with process metrics.

Table 1 Dataset Description

AEEEM Dataset	Total Features	Total Samples	Defect Artifact	Non-Defect Artifact
JDT	61	997	206	791
MyLyn	61	1862	245	1617

5. RESULTS AND DISCUSSION

5.1. Classification Accuracy Analysis

Line of Code, Cyclomatic complexity, Halstead Complexity and McCabe complexity metrics for each module is evaluated from the defect dataset. Enhanced GA is used for feature selection; out of 61 features, the reliable ones from the dataset are filtered. The Support Vector Machine without ensemble GA gives an accuracy of 50.83%, and 48.83% seems to be a lesser accuracy. The enhanced GA filters the optimistic feature, and GWO tunes the hyperparameter γ. The SSVM model integrates the power of GA and GWO to come out with better accuracy. The hyperparameter gets the best-fit value by adapting the inspirational hunting phenomena of Grey Wolf α. Ten-fold cross-validation is performed on the dataset and classified using SVM with gamma γ= 0.1 and C =100. The gamma parameter is set to a different range from 0.1 to 1. Better accuracy is obtained as 75.30 % in the case of JDT and 77.30 % for MyLyn dataset setting α parameter. A confusion matrix is presented in Table 2 that assesses the SSVM model's performance. The matrix compares the SVM, SVM-GA and SSVM models. Henceforth, the findings are displayed below. The significant improvement attained by the SSVM model highlights the efficiency in predicting software defects.

Table 2 Classification Accuracy Analysis for SSVM

Metrics	JDT			MyLyn		
	SVM	SVM-GA	SSVM	SVM	SVM-GA	SSVM
TPR	51.74	65.14	77.20	47.75	61.08	81.17
TNR	49.92	64.87	73.39	49.92	64.87	73.39
Classification Accuracy	50.83	65.00	**75.30**	48.83	63.00	**77.30**

The TRP, TNR and Classification Accuracy for the SSVM model is shown in Figure 2, with 75.30 % in the case of JDT and 77.30 % for MyLyn dataset.

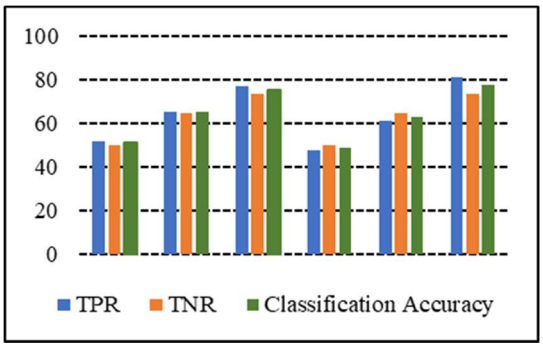

Figure 2 Classification Accuracy Analysis on JDT and MyLyn Data

5.2. F1 Score Analysis

The F1 score gives a balanced measurement of both precision and recall measures. It is the harmonic mean of precision and recall. Beforehand the precision for any defect module that is projected to be positive, precision is the percentage of correctly predicted positive instances. Precision is calculated below as in Eq. (16)

$$Precison = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (16)$$

Recall counts how many positive instances were properly predicted out of all the positive instances. The metric is calculated as in Eq (17)

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (17)$$

F1 metric significantly doubles the product of the correctly predicted positive value and the count of actual positive instances from the positive instances. The value is calculated from Eq (18)

$$F1\ Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (18)$$

It is observed from the below Figure 3 that the convergence of the F1 Score for the model SSVM over the SVM, SVM-GA has significantly improved. A tabulated view of the F1 Score comparison is presented in Table 3, which exhibits a 75.88 % increase in accuracy for JDT data and 78.26 % for the MyLyn data. Enhancement of the SVM model by hybridizing GA and GWO has proved to be imminent.

Table 3 F1 Score Analysis for SSVM

Classifiers	F1 Score	
	JDT	MyLyn
SVM	51.32	48.31
SVM-GA	64.73	61.95
SSVM	75.88	78.26

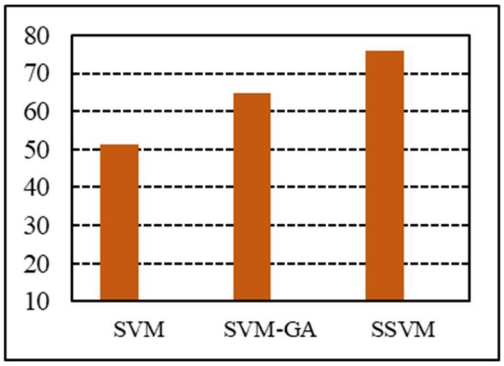

Figure 3a F1 Score Analysis of SSVM Model on JDT Data

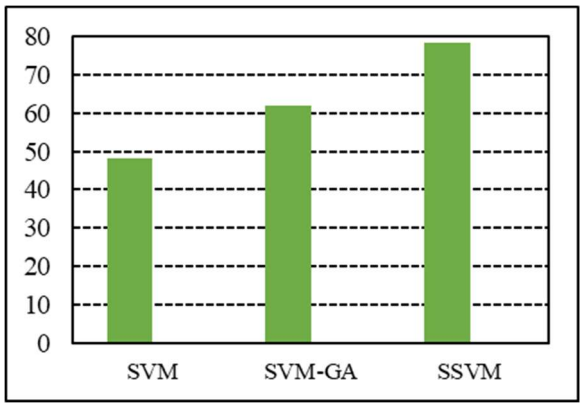

Figure 3b : F1 Score Analysis of SSVM Model on MyLyn Data

6. CONCLUSION

Software defect prediction is necessary for any software's quality improvement. This paper proposes a Sophisticated Support Vector Machine (SSVM) model that combines the powerful features of Genetic Algorithm and Grey Wolf Optimization. An investigation is done on the dataset to address the representativeness issue, and appropriate defect data is selected. The proposed SSVM model is inspected on the defect data from JDT and MyLyn Java Projects from AEEEM repository. A genetic

algorithm is involved in the feature selection process. A support Vector Machine is used to train the model with the defect data. The prediction of faulty and non-faulty classification of modules is improvised by using a GWO algorithm that finds the optimum fitness value of γ tuning hyperparameter. The performance of SSVM is measured using True Positive Rate, False Positive Rate, Confusion Matrix and F1 Score. The experimental results show a higher accuracy rate of 75.30% and 77.30% comparing the traditional SVM and SVM – GA. The challenge of dataset representatives has been achieved by proposing the SSVM model. A temporal dynamic problem still exists, which significantly affects the performance of prediction models that will be discussed in future work.

REFERENCES:

[1] Ceran, A. A., Ar, Y., Tanrıöver, Ö. Ö., & Ceran, S. S. (2023). Prediction of software quality with Machine Learning-Based ensemble methods. Materials Today: Proceedings, 81, 18-25.

[2] Son, T. T., Lee, C., Le-Minh, H., Aslam, N., & Dat, V. C. (2022). An enhancement for image-based malware classification using machine learning with low dimension normalized input images. Journal of Information Security and Applications, 69, 103308.

[3] Liu, X., Miramini, S., Patel, M., Ebeling, P., Liao, J., & Zhang, L. (2023). Development of numerical model-based machine learning algorithms for different healing stages of distal radius fracture healing. Computer Methods and Programs in Biomedicine, 233, 107464.

[4] Cai, Y., Xiao, L., Kazman, R., Mo, R., & Feng, Q. (2018). Design rule spaces: A new model for representing and analyzing software architecture. IEEE Transactions on Software Engineering, 45(7), 657-682.

[5] Eken, B., & Tosun, A. (2021). Investigating the performance of personalized models for software defect prediction. Journal of Systems and Software, 181, 111038.

[6] Yu, T., Wen, W., Han, X., & Hayes, J. H. (2018). Conpredictor: Concurrency defect prediction in real-world applications. IEEE Transactions on Software Engineering, 45(6), 558-575.

[7] Rathaur, S., Kamath, N., & Ghanekar, U. (2020, July). Software defect density prediction based on multiple linear regression. In 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA) (pp. 434-439). IEEE.

[8] Ashtari, A., & Alizadeh, B. (2022). A comparative study of machine learning classifiers for secure RF-PUF-based authentication in internet of things. Microprocessors and Microsystems, 93, 104600.

[9] Gomes, L., da Silva Torres, R., & Côrtes, M. L. (2023). BERT-and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: a comparative study. Information and Software Technology, 160, 107217.

[10] Pushphavathi, T. P. (2017, August). An approach for software defect prediction by combined soft computing. In 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS) (pp. 3003-3006). IEEE.

[11] Yu, X., Rao, J., Hu, W., Keung, J., Zhou, J., & Xiang, J. (2023). Improving effort-aware defect prediction by directly learning to rank software modules. Information and Software Technology, 107250.

[12] Ma, Y., Mockus, A., Zaretzki, R., Bradley, R., & Bichescu, B. (2020). A methodology for analyzing uptake of software technologies among developers. IEEE Transactions on Software Engineering, 48(2), 485-501.

[13] Sharma, U., & Sadam, R. (2023). How far does the predictive decision impact the software project? The cost, service time, and failure analysis from a cross-project defect prediction model. Journal of Systems and Software, 195, 111522.

[14] Bai, J., Jia, J., & Capretz, L. F. (2022). A three-stage transfer learning framework for multi-source cross-project software defect prediction. Information and Software Technology, 150, 106985.

[15] Chennappan, R. (2023). An automated software failure prediction technique using hybrid machine learning algorithms. Journal of Engineering Research, 11(1), 100002.

[16] Qu, Y., Li, Z., Zhao, J., & Li, H. (2022, October). Unbalanced data processing for software defect prediction. In 2022 4th International Conference on Data-driven Optimization of Complex Systems (DOCS) (pp. 1-6). IEEE.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[17] Tong, H., Lu, W., Xing, W., & Wang, S. (2023). ARRAY: Adaptive triple feature-weighted transfer Naive Bayes for cross-project defect prediction. Journal of Systems and Software, 202, 111721.

[18] Anyanwu, G. O., Nwakanma, C. I., Lee, J. M., & Kim, D. S. (2023). RBF-SVM kernel-based model for detecting DDoS attacks in SDN integrated vehicular network. Ad Hoc Networks, 140, 103026.

[19] Manchala, P., & Bisi, M. (2022). Diversity based imbalance learning approach for software fault prediction using machine learning models. Applied Soft Computing, 124, 109069.

[20] Saravanan, P., & Sangeetha, V. (2022). African buffalo optimized multinomial softmax regression based convolutional deep neural network for software fault prediction. Materials Today: Proceedings, 61, 619-626.

[21] Jagtap, M., Katragadda, P., & Satelkar, P. (2022, January). Software Reliability: Development of Software Defect Prediction Models Using Advanced Techniques. In 2022 Annual Reliability and Maintainability Symposium (RAMS) (pp. 1-7). IEEE.

[22] Khatri, Y., & Singh, S. K. (2023). An effective software cross-project fault prediction model for quality improvement. Science of Computer Programming, 226, 102918.