International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V72I9P117

Volume 72 Issue 9, 203-219, September 2024

Original Article

Clown Fish Optimized – Modified Support Vector Machine (CFO-MSVM) for Software Defect Prediction

Medhunhashini D. R.[1], KS Jeen Marseline[2]

[1,2]*Department of IT and Cognitive Systems, Sri Krishna Arts and Science College, Tamilnadu, India.*

[1]*Corresponding Author : medhun.hashini@gmail.com*

Received: 15 May 2024 Revised: 08 August 2024 Accepted: 17 August 2024 Published: 28 September 2024

Abstract *- The growing software industry and the necessity for software development has increased rapidly. The biggest challenge is to develop software in minimal time with fewer resources and bug-free. Software defect prediction has the privilege of predicting the software bug at the earliest to avoid chaos. This paper presents a novel method combining the nature-inspired optimization technique with the Support Vector Machine, proposing a Clown Fish Optimized – Modified Support Vector Machine (CFO-MSVM) classifier for earlier effective classification of the bug. The objective function of the proposed classifier is to tune the hyperparameter of SVM following the swarm intelligence of the clown fish crowd. The Java Developers Toolkit (JDT) dataset from AEEEM repository is used as the bench marker to validate the CFO-MSVM classifier. The classifier is investigated using a grid search for the regularization parameter C, and the number of iterations is set to 100. Precision, Recall and F Score Metrics are used for evaluation. FMI and MCC statistical measurements are employed to define accuracy further. The CFO-MSVM classifier segregates the defect and non-defect modules with 87.32 % accuracy compared to the existent SVM and SVM-GA classifiers, which have 50.83% and 65.00%, respectively.*

Keywords *- Accuracy, Clown Fish Optimization, Software defects, Support Vector Machine, Tuning parameters.*

1. Introduction

Software Defect Prediction (SDP) is a critical discipline within software engineering, as it plays a pivotal role in forecasting and preventing defects in software systems. By actively analyzing and interpreting software metrics, developers can spot the code boundary that will have defects. Through the utilization of historical data and the application of algorithms and statistical models, SDP unveils patterns and factors associated with the occurrence of defects, enabling proactive measures to be taken. The primary objective of SDP is to forecast potential defects early in the software development lifecycle[1]. By leveraging historical data, such as previous defect reports and bug fixes, and employing machine learning algorithms, defect prediction models learn from past experiences and extract valuable insights. These models capture patterns, relationships, and correlations between various software metrics and the occurrence of defects. Examples of such metrics include code complexity, code churn, and developer experience. The core software area were defects are found the most is analyzed [2]. The proactive nature of SDP provides developers with a significant advantage. Instead of relying solely on reactive defect detection and correction, teams can take immediate action based on the predictions made by the models. Developers can allocate resources effectively by identifying high-risk areas early on, prioritizing testing efforts, and implementing preventive measures. This approach minimizes the effort and cost associated with defect detection and correction, leading to improved software quality and reduced development cycles. Efficient resource allocation is one of the critical benefits of SDP[3]. The code developers have resource constraints, especially regarding time, budget, and human needs. By leveraging defect prediction models, teams can strategically allocate their resources to focus on the areas of the codebase that are more likely to contain defects. This targeted resource allocation ensures that critical components receive the necessary attention and resources, reducing the risk of undetected defects[4]. SDP allows for the prioritization of testing efforts. Testing is a process involving the most time and resources during the SDLC. By utilizing defect prediction models, teams can prioritize their testing efforts on the areas identified as high-risk[5]. This proactive method makes sure that the software is cleanly tested and increases the likelihood of detecting and resolving potential issues before the software is released. By focusing testing efforts on high-risk areas, teams can achieve more effective defect detection, resulting in improved software quality and customer satisfaction. In addition to resource allocation and testing prioritization, SDP improves software quality. By identifying potential problem areas before defects manifest, developers can implement preventive measures such as code refactoring, process improvements, or additional quality assurance activities. This

proactive approach addresses potential issues early in the development process, reducing the likelihood of defects and enhancing the overall software quality[6]. Most SDP models do not record proper defects as they depend on the past defect history, several assumption ideas and correlations between the possible sources of defects. The quality and significance of the past defect data are to be highly consistent as they decide the quality factor of any software. As the need for the software gets updated and the new software evolves consistently, the software systems of the SDP must be effective [7]. New evolved mental methods for predictions are required as any change or update to the codebase may introduce new defects. SDP is a valuable discipline in software engineering those aids in forecasting and preventing defects in software systems. Defect prediction models provide developers with valuable insights into high-risk areas by actively analyzing software metrics, identifying patterns, and leveraging historical data. This enables efficient resource allocation, prioritization of testing efforts, and the implementation of preventive measures[8].SDP improves software quality, reduces development cycles, and enhances customer satisfaction.

1.1. Machine Learning

Machine Learning (ML) has significantly impacted the field of SDP, introduced new capabilities and improved the accuracy of predictions. ML enhances defect detection and enables proactive defect prevention through its advanced algorithms and data analysis techniques. ML in SDP is its ability to identify patterns and relationships in historical data[9]. By analyzing past defect reports, bug fixes, and other relevant data, ML models can uncover hidden patterns and factors contributing to defects. These patterns may include specific coding practices, software metrics, or environmental factors associated with higher defect rates[10]. By learning from historical data, ML models can predict which areas of the codebase are more likely to contain defects, enabling developers to prioritize testing and allocate resources accordingly. ML algorithms also excel at handling complex and non-linear relationships. Unlike traditional statistical methods, ML models can capture intricate interactions between software metrics and defects. This allows for a more comprehensive understanding of how different factors influence the occurrence of defects. For example, ML algorithms can detect subtle correlations between code complexity, code churn, and the likelihood of defects. By considering multiple metrics simultaneously, ML models can provide more accurate predictions and help developers focus their efforts on the most critical areas[11]. ML enables the continuous improvement of defect prediction models. Models can be updated and maintained to reflect new knowledge. It is essential to be available by retraining them with it. In the ever-evolving world of software development, new coding conventions, technologies, and fault sources appear on a regular basis, and this flexibility is essential. Developers can ensure that their defect prediction efforts remain relevant and aligned with the evolving software landscape by continuously updating the models[12]. The ML models integrate different data sources for software defect prediction. Along with Code-related metrics, some data from bug-tracking systems, version control systems, code review comments, and user feedback are taken for ML classifiers. This integration allows for a meaningful study of the software development process and provides a broader context for defect prediction[13].

By considering multiple data streams, ML models can capture a more holistic view of the factors that influence the occurrence of defects, resulting in more accurate and reliable predictions. The quality and relevance of the training data, the selection of appropriate features, and the use of robust evaluation techniques are crucial to ensure the reliability and effectiveness of the models. Additionally, the interpretability of ML models remains a challenge. While these models can provide accurate predictions, understanding the underlying reasons for their predictions can be complex. Efforts are being made to develop interpretable ML methods to shed light on the characteristics driving the fault predictions, upgrading transparency and trust in the defect prediction process[14].

1.2. Problem Statement

Data biases in SDP datasets are a significant challenge that undermines the reliability and fairness of prediction models. These biases, stemming from factors like defect report selection criteria, reporting culture, or focus on specific defect types, lead to skewed representations of defect occurrences and hinder the models' ability to capture true patterns and contextual factors. To address this problem, it is crucial to develop methodologies that identify and mitigate biases, ensuring unbiased and representative datasets. The accuracy and effectiveness of defect prediction models ultimately improve software quality and reduce maintenance efforts. This requires careful examination of data collection processes, implementing strategies to address reporting biases and employing techniques to balance the representation of defect types and contexts in the dataset.

1.3. Motivation

The motivation for the research is addressing data biases in SDP, which lies in the potential to improve software quality, reduce maintenance efforts, promote fairness, and enhance transparency and trust in defect prediction models. By developing methodologies to identify and mitigate biases, researchers can enable early detection and prevention of software defects, leading to significant time and resource savings. Addressing biases ensures fair representation of all modules, versions, and contexts, promoting equity in defect prediction. Furthermore, uncovering and mitigating biases enhances the interpretability and explainability of models, fostering transparency and trust in the predictions and making them more actionable in real-world software development scenarios. This research has the potential to advance the field of software engineering and contribute to the overall improvement of software development processes.

1.4. Objective

The study aims for ML algorithms that effectively address the challenge of data biases in SDP. One of the data biases generally includes the selection of data for training the model. The research aims to design novel methodologies that can identify and mitigate selection biases in defect prediction datasets, ensuring the prediction models' reliability, fairness, and generalizability. The specific research objectives include:

- Designing robust algorithms: Develop ML algorithms that are resilient to data biases and can effectively handle imbalanced and biased datasets commonly encountered in SDP.
- Bias detection and mitigation: Develop techniques to identify and quantify biases present in defect prediction datasets, enabling researchers to understand the nature and extent of the biases.
- Bias mitigation strategies: Propose strategies to mitigate biases in the datasets, like data preprocessing techniques, sampling methods, and hyperparameter adjustments, ensuring a fair representation of defect occurrences across different modules, versions, and contextual factors.
- Performance evaluation and comparison: Conduct extensive experimental evaluations to assess the working of the proposed algorithms against existing approaches, considering various metrics like accuracy, precision, recall, and fairness measures to demonstrate the effectiveness of the proposed methodologies in addressing data biases.

By achieving these research objectives, this research aims to advance ML techniques in SDP, improving the prediction models' accuracy, fairness, and applicability. The research findings will provide valuable insights and guidance for practitioners in effectively handling data biases and deploying reliable defect prediction systems in real-world software development environments.

2. Literature Review

"Artificial Immune Systems in cross-project software fault prediction"[15] works on the mammalian immune patterns swiftly to address the defect prediction problem. The investigation was performed on the java projects from the PROMISE defect repository, focusing on the immunological system. Based on the Friedman and Nemenyi post-hoc test summary, the Immunos-1 and Immunos-99 performed better in reference to the Recall measure. The results of the Wilcoxon test point to the need for researchers working on intra-project defect prediction issues to assess their models for inter-release setups."Salp Swarm Optimizer (SSO) for modeling the software fault prediction model"[16] encompasses a combination with Backpropagation Neural Network (BPNN) to anticipate the defect prediction problem. The hyperparameter selection and tuning have happened by combining the SSA optimizer and BPNN to enhance prediction. The dataset used validates performance measures such as AUC, Sensitivity, Specificity, Accuracy, Error Rate, and Confusion Matrix. The results show that the combined SSA-BPNN outperforms other conventional methods better. Hybrid use of algorithms has higher prediction accuracy in defect prediction. "Software Metrics and Fault Prediction model"[17] to find the defect sets employs a framework to authenticate the validity of the metric-based source code. The model aims to find and reduce the features that are not relevant; hence, to improve prediction performance, t-test analysis and univariate logistic regression analysis are performed on the metric to predict the defect module. A correlation analysis is set to find the relationship between the fault code using metrics. The experiment was conducted on fifty-six java projects. The results revealed that the validation framework has considerably improved its efficiency by a threshold value of low – 48.89%, median – 39.26%, and high – 27.86%. "Threshold calculation techniques to find fault-module"[18] framework primarily investigated three threshold techniques: ROC Curve, VARL (Value of an Acceptable Risk Level) and Alves Ranking with four machine learning-based models and two clustering-dependent prediction models. Datasets from the PROMISE repository and Eclipse project were used for investigation. ROC curve performed best compared to other threshold evaluators, such as Alves Ranking and VARL method. The experiment concluded that the better threshold measure for the fault prediction methods can be ROC.

"ACO-based feature weighting method"[19] for software defect prediction tends to find the severity of the bug that tends to deliver the software late. This work proposes ant colony optimization to select the relevant features for classification. A combined Ant Colony Optimization with Naïve Bayes, Support Vector Machine, DeepFM and F-Support Vector Machine to classify the defects into multiple classes. The bug severity data is collected from Eclipse, Mozilla, OpenFOAM, JBoss and Firefox. The usefulness of the technique was measured using accuracy, precision, recall, and F measures. For five benchmark projects, the accuracy score of the ACO-F-SVM, ACO-NB, ACO-SVM, ACO-DeepFM, NB, SVM, F-SVM, and DeepFM approaches range from 85.73 to 89.38%, 78% to 80%, 73% to 76%, 92.67% to 97.27%, 71% to 77%, 65% to 74%, 78.21% to 81.28%, and 90.02% to 95.2%. "AI-based software bug assessment model" [20] helps in managing the bug repository smartly combining Software Bug Triaging (SBT) techniques and AI. A systematic review of the bug reporting item analysis was carried out using PRISMA. Around 123 samples were taken up for AI study and implementation. AI-biased risk computations were carried out using the Cochrane protocol. A deep learning approach has shown elevations in learning capacity, better scalability, and better performance than traditional methods. The AI-SBT framework efficacy was measured using accuracy, mean, precision and recall metrics. "Unsupervised defect prediction model for software faults" applies clustering models on the unlabeled dataset using CUDP. The dataset used for the study includes 27 versions of a project with three unlabeled features.

The experimental results showed that the CUDP model moderately performs defect prediction under the hybrid clustering methods. "Nonlinear Manifold Detection Techniques"[21] have been proposed to identify and eliminate immaterial features to improve the prediction strategy little in a higher rate. The method employs dimensionality reduction for more precise and accuracy. An innovative approach to achieve the objective is to create a new model based on nonlinear MDTs and evaluate its performance against current feature selection methods to determine the most precise defect prediction strategy. With the aid of the Friedman test and Post Hoc analysis, the effectiveness of various classification methods employing both new and established methodologies have been assessed, contrasted, and statistically tested. The findings demonstrated that nonlinear MDT is more performance-oriented than all other methodologies combined in terms of accuracy.

"COMET for Software Defect Prediction"[2] employs finding the defect module components to increase software quality. Many functional dependencies tend the COMET to implement coupling metrics to improve prediction performance. Conceptual coupling finds a logical code similar to the source code. COMET, a conceptual coupling metric experiment, was conducted on the public dataset using both supervised and unsupervised ML models. "RSMOTE-based Data Imbalance Processing (RDIP)"[22] aims to frame a machine learning model to solve the defect class imbalance problem in the defect dataset for defect class classification. The outlier data is removed from the dataset. The Computational Class Fuzzy Algorithm (FCMD) calculates the fuzzy membership and fuzzy labels of each point after the normalization of the outlier data, calculating the European distance between points in the data noise reduction process, which removes the hazard points and noise points in accordance with the selection Boundary Point Algorithm (BRS). The experiments were conducted on the NASA promise defect prediction dataset. The F1 measure was identified to be higher at 6.98% compared to other algorithms. Bio-inspired optimization is significant in all major research types [23-32].

"Smell-based defect prediction model"[33] encompasses a prediction strategy using machine learning algorithms by learning features that have shown better accuracy in the prediction of code smells. Defect prediction studies have not covered the design code smells that avoid object-oriented principles. This model is studied on 97 projects to find the performance of prediction techniques with several classifiers. Traditional smells from the literature employing design code smells as features are considered. The performance of the models based on the categories of design code smells is grouped and examined, and finally, an improvement of 4.1% for the AUC score over the models trained is achieved. Design smells are, therefore, a useful supplement to the scents frequently investigated in the literature for defect prediction. "Transfer Learning Method for Software Defect Prediction"[34] for multi-source defect data encounters a careful mechanism for source selection from multiple projects. The work proposes a transfer methodology to reduce the difference peripheral differences. Four multi-source utilization schemes and five source selection techniques are created, and by weighing their effects on prediction performance, the optimal one to be employed in stages 1 and 3 of the 3SW-MSTL is selected. Next, the data from 30 commonly used open-source projects were evaluated to measure the performance of 3SW-MSTL with four multi-source and six single-source CPDP methods, a baseline Within-Project Defect Prediction (WPDP) method, and two unsupervised methods.

"DNP using Regression Learning" [35] proposes the effective use of regression algorithms with resampling techniques. The study also used ensemble learning techniques and optimized feature selection for feasible error detection. The experiments were conducted on 18 PROMISE datasets. The average absolute error and high pred (0.3) were inspected to find the performance. "Improving Defect Prediction Efficiency using Decision Tree and Bayesian" helps in improving the classification accuracy for software defect prediction. The accuracy metrics over the PROMISE dataset have proved that the Decision Tree has better classification than the Bayesian classifier, ultimately building software with high quality. "Neural Networks based Defect classification" [36] proposes a hybrid deep residual neural network method that combines well-established computer vision methods for defect segmentation and deep residual neural networks with grid search-based hyperparameter optimization for defect classification. The designed model is compared using metrics F1 score, Cohen's Kappa Coefficient and Confusion matrix to check computing capability. The results show that the suggested hybrid technique while requiring the least amount of computational time, offers the best defect classification of defects in semiconductor wafers in terms of F1-score (99.443%).

3. Clown Fish Optimized – Modified Support Vector Machine (CFO-MSVM)

The Clown Fish Optimization (CFO) algorithm solves the problems of optimality by simulating the movement of schools of fish and the intelligence underlying these behaviors.

3.1. Initialization of CFO Parameters

The CFO-MSVM algorithm commences with the vital step of initializing parameters specific to the Fish Swarm Optimization (CFO) component. These parameters govern the behavior of the fish swarm, influencing their movement and exploration within the search space. Let S represent the swarm size, t_{max} denote the maximum number of iterations and $w_{max}$and w_{min} signify the maximum and minimum inertia

weights, respectively, as in Equation (1). The initialization process ensures that the fish swarm operates within defined bounds throughout the optimization process.

$$S, t_{max}, w_{max}, w_{min} \geq 0 \quad (1)$$

In conjunction with the CFO, a Modified Support Vector Machine (MSVM) requires its parameters for feature modification. Let C denote the regularization parameter, γ represent the kernel coefficient, ϵ signify the tube width, and Δ indicate the feature modification parameter as in Eq (2). The initialization of these parameters is crucial for shaping the modified features derived from MSVM.

$$C, \gamma, \epsilon, \Delta \geq 0 \quad (2)$$

Define the training dataset as D_{train}, consisting of labelled instances x_i and their corresponding classes y_i. This initialization ensures the availability of a structured dataset for the subsequent training and optimization processes, as in Equation (3).

$$D_{train} = \{(x_i, y_i)\}, \quad i = 1,2, \dots, n \quad (3)$$

Formulate the objective function (f_{obj}) that combines both the MSVM and CFO objectives. The objective function guides the optimization process, steering the fish swarm towards optimal solutions. Let θ represent the parameters to be optimized as in Equation (4).

$$f_{obj}(\theta) = f_{MSVM}(\theta) + f_{CFO}(\theta) \quad (4)$$

Initialize the modified features (X_{mod}) derived from MSVM, incorporating the feature adjustment parameter Δ as in Equation (5). This initialization sets the foundation for subsequent optimization steps.

$$X_{mod} = MSVM\_ModifyFeatures(X, \Delta) \quad (5)$$

Identify the best global position (G_{best}) within the CFO swarm based on the fitness evaluation using the objective function. The best global position influences the movement of the entire fish swarm, as shown in Equation (6).

$$G_{best} = \arg min_{\theta} f_{obj}(\theta) \quad (6)$$

The initialization phase of the CFO-MSVM algorithm establishes a robust foundation for subsequent optimization and training processes. The defined parameters and objectives guide the fish swarm and MSVM feature modification, ensuring a systematic and well-structured approach to enhancing classification accuracy.

3.2. Feature Modification Process

The Feature Modification step in CFO-MSVM plays a pivotal role in enhancing the discriminatory power of features. It involves adjusting the original feature set X to X_{mod} using the modified feature adjustment parameter Δ as in Equation (7). The modification aims to improve the separability of classes, contributing to the overall effectiveness of the classification process.

$$X_{mod} = MSVM\_ModifyFeatures(X, \Delta, \gamma) \quad (7)$$

The feature modification leverages a kernel-based transformation facilitated by the parameter γ. The kernel coefficient γ influences the shape and flexibility of the transformation, allowing for the creation of non-linear decision boundaries as in Equation (8). The kernel function K operates on pairs of data points, effectively mapping the input features to a higher-dimensional space.

$$K(x_i, x_j) = exp\left(-\gamma \|x_i - x_j\|^2\right) \quad (8)$$

Employing MSVM involves optimizing the parameters C, γ, and ϵ to attain a hyperplane that maximally separates classes in the modified feature space. The decision function $f(x)$ is defined based on the support vectors (SV) and their corresponding coefficients (α) as in Equation (9).

$$f(x) = \sum_{i=1}^{n_{SV}} \alpha_i K(x, x_i) + b \quad (9)$$

The feature adjustment parameter Δ directly influences the extent of modification applied to the original features. Its optimization is integral to achieving an optimal balance between feature enhancement and preserving discriminative information. Equation (10) makes the adjustment based on the performance feedback from the CFO component.

$$\Delta = \Delta - \eta \frac{\partial f_{obj}}{\partial \Delta} \quad (10)$$

Define an optimization objective function for the feature modification process. This function combines the MSVM classification objective with the feedback from the CFO component. It encapsulates the dual goals of achieving high classification accuracy and guiding the fish swarm towards optimal solutions, as in Equation (11).

$$f_{MSVM}(\theta) = MSVM\_Objective(\theta) \quad (11)$$

The Feature Modification step in CFO-MSVM intricately weaves together the principles of MSVM and the guidance provided by the CFO. Through kernel-based transformations and parameter adjustments, this step ensures that the modified features contribute significantly to the discrimination of classes. The synergy between MSVM and CFO creates a dynamic and adaptive feature modification process that adapts to the evolving optimization landscape.

3.3. Initialization of Fish Swarm Positions

The CFO-MSVM initiates the CFO swarm by randomly distributing fish individuals in the solution space. Each fish represents a potential solution configuration for the MSVM parameters. The position as in Equation (12) for each fish F_i is a vector P_{os_i} in the N-dimensional solution space.

$$P_{os_i} = [x_{i1}, x_{i2}, \dots, x_{iN}] \quad (12)$$

For each fish in the swarm, the objective function f_{CFO} is computed based on the corresponding MSVM parameters. The objective function reflects the collective behavior of the fish swarm and guides the optimization process towards configurations that enhance classification accuracy while considering the CFO principles as in Equation (13).

$$f_{CFO}(Pos_i) = EvaluateObjective(Pos_i) \quad (13)$$

The movement of each fish is governed by a set of rules that emulate the principles of fish swarm behavior. The rules incorporate elements of exploration and exploitation, ensuring a balanced exploration of the solution space while converging towards the neighborhoods as in Equation (14). Both individual and collective influences define the fish movement.

$$Move_i(t) = IndividualInfluence_i(t) + CollectiveInfluence_i(t) \quad (14)$$

The individual influence component guides each fish based on its historical movement patterns. It encourages exploration by allowing fish to move towards unexplored regions. The p parameter determines the impact of the individual influence component on the fish's movement, as in Equation (15).

$$IndividualInfluence_i(t) = p.\left(Pos_{best,i}(t) - Pos_i(t-1)\right) \quad (15)$$

The collective influence component represents the impact of the neighboring fish on the movement of a particular fish. It fosters convergence towards promising regions by aligning the movement of neighboring fish. The q parameter governs the strength of this collective influence as in Equation (16).

$$CollectiveInfluence_i(t) = q.\sum_{j=1}^{N_{neighbors}} \left(Pos_{best,i}(t) - Pos_i(t-1)\right) \quad (16)$$

The final step involves updating the positions of the fish based on the calculated movement. The new positions reflect the fish swarm's collective exploration and exploitation strategies as in Equation (17). This iterative process continues until a convergence criterion is met.

$$Pos_i(t) = Pos_i(t-1) + Move_i(t) \quad (17)$$

The CFO swarm initialization sets the stage for the dynamic optimization process in CFO-MSVM. By aligning with fish swarm principles, the CFO component ensures a balanced exploration-exploitation trade-off, leading to the discovery of optimal configurations for the MSVM parameters.

3.4. Objective Function

The heart of the CFO-MSVM optimization process lies in the objective function f_{CFO}, which quantifies the performance of the MSVM parameters. This function integrates the SVM classification accuracy and the CFO-inspired exploration-exploitation principles, creating a holistic measure of solution quality as in Equation (18).

$$f_{CFO}(Pos_i) = \frac{1}{N}\sum_{i=1}^{N} EvaluateObjective(Pos_i) \quad (18)$$

The SVM classification accuracy component represents the conventional evaluation metric for MSVM. It assesses the performance of the MSVM parameters in terms of correctly classified instances. Equation (19) calculates the accuracy Acc_i based on the confusion matrix.

$$Acc_i = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \quad (19)$$

The CFO exploration component captures the exploration aspect inspired by fish swarm behaviour. It leverages the objective function's historical performance to encourage fish to explore regions with potential improvement. The p parameter modulates the impact of this exploration component as in Equation (20).

$$Exploration_i = p.\left(\frac{f_{CFO}\left(Pos_i(t-1)\right) - Min_{History}}{Max_{History} - Min_{History}}\right) \quad (20)$$

Conversely, the CFO exploitation component focuses on exploiting well-performing regions. It encourages fish to use areas that historically yield better objective function values. The q parameter regulates the influence of this exploitation component as in Equation (21).

$$Exploitation_i = q.\left(\frac{Max_{History} - f_{CFO}\left(Pos_i(t-1)\right)}{Max_{History} - Min_{History}}\right) \quad (21)$$

The objective function components, SVM classification accuracy, exploration, and exploitation, are combined to create a measure of solution quality. The final objective function f_{CFO} balances the traditional performance evaluation with CFO-inspired strategies as in Equation (22).

$$f_{CFO}(Pos_i) = Acc_i + Exploration_i + Exploitation_i \quad (22)$$

The objective function is evaluated for each fish in the CFO swarm. The calculated objective function values guide the movement of the fish swarm, directing them towards regions that offer the potential for improved MSVM parameter configurations. This iterative process ensures continuous refinement of the objective function and drives the optimization towards optimal solutions for MSVM parameters in the CFO-MSVM framework.

3.5. Initialization of Global Best Position

The global best position, denoted as $Pos_{GlobalBest}$, serves as a pivotal reference point in the CFO-MSVM optimization. This step initializes this position, establishing the starting point for the optimization process. The global best

position tracks the fish swarm's historical performance, guiding the optimization towards regions that exhibit superior MSVM parameter configurations.

$$Pos_{GlobalBest}(t) = GetInitialGlobalBest() \quad (23)$$

The initial evaluation of the global best position involves assessing its objective function value, denoted as $f_{GlobalBest}(t)$. This value represents the performance of the MSVM parameters associated with the global best position. It provides a benchmark for comparison as the optimization progresses.

$$Pos_{GlobalBest}(t) = f_{CFO}\left(Pos_{GlobalBest}(t)\right) \quad (24)$$

The fish swarm undergoes individual evaluations and each fish's objective function value $\left(f_{CFO}(Pos_i(t))\right)$ is calculated. The comparison between the fish's objective function value and the global best position's objective function value determines whether the fish needs to update the global best position.

$$f_{CFO}(Pos_i(t)) < f_{GlobalBest}(t) \rightarrow UpdateGlobalBest(Pos_i(t)) \quad (25)$$

The update rule for the global best position involves replacing the current global best position with the fish's position if the fish exhibits a superior objective function value. This mechanism ensures that the global best position continuously reflects the MSVM parameters associated with the best-performing solution in the fish swarm.

$$Pos_{GlobalBest}(t) = Pos_i(t) \quad (26)$$

The CFO-MSVM framework incorporates adaptive adjustment mechanisms for the exploration (p) and exploitation (q) parameters. These adjustments make the algorithm work beyond the optimization landscape, balancing exploration and exploitation based on the swarm's performance.

$$p(t+1) = AdaptParameter(p(t), f_{GlobalBest}(t))$$
$$q(t+1) = AdaptParameter(q(t), f_{GlobalBest}(t)) \quad (27)$$

The initialization of the global best position, along with the subsequent evaluations and updates, marks the commencement of a highly defined solution space. This is the basic for the fish swarm to collectively investigate and use the solution space, guided by the evolving global best position. The adaptive adjustment of parameters ensures the algorithm's responsiveness to the optimization, facilitating continuous improvement in the search for optimal MSVM parameter configurations within the CFO-MSVM framework.

3.6. Position Update for Each Fish

The swarm movement in CFO-MSVM involves updating the position of each fish in the fish swarm. This step ensures that each fish explores the solution space, guided by its movement. The position update equation captures the dynamic movement of each fish towards potential regions of interest.

$$Pos_i(t+1) = UpdatePosition(Pos_i(t), p(t), q(t)) \quad (28)$$

The exploration (p) and exploitation (q) factors are crucial in determining the extent of a fish's exploration and exploitation movements. These factors dynamically adjust based on the global best position's performance, influencing the overall swarm movement. The exploration factor enhances exploration during the early stages, while the exploitation factor intensifies exploitation as the optimization progresses.

$$p(t+1) = AdaptActor(p(t), f_{GlobalBest}(t))$$
$$q(t+1) = AdaptFactor(q(t), f_{GlobalBest}(t)) \quad (29)$$

The movement of each fish towards the global best position represents a collective effort to converge towards a promising solution. The movement equation incorporates the influence of both investigator and Exploit factors, making a stability finding the diverse areas and exploiting the potential of the current best solution.

$$Pos_i(t+1) = MoveTowardsGlobalBest\left(\begin{matrix} Pos_i(t), \\ Pos_{GlobalBest}(t), p(t), \end{matrix}\right. \quad (30)$$

Swarm movement involves the coordination of multiple fish to explore the solution space collectively. The swarm coordination mechanism ensures that individual fish movements contribute synergistically to the exploration and exploitation efforts. It prevents excessive exploration or exploitation by coordinating the movements of the entire fish swarm.

$$Pos_{Swarm}(t+1) = CoordinateSwarm(Pos_i(t+1), Pos_{Swarm}(t)) \quad (31)$$

The iterative nature of swarm movement characterizes the continuous exploration and exploitation dynamics within CFO-MSVM. As each fish updates its position based on the adaptive exploration and exploitation factors, the swarm collectively progresses towards potential optimal solutions. This iterative movement ensures that the fish swarm adapts to the changing optimization landscape, dynamically exploring and exploiting the solution space to enhance the search for optimal MSVM parameter configurations.

3.7. Feature Adjustment Mechanism

In CFO-MSVM, the feature adjustment process aims to optimize the feature set for improved classification accuracy. The feature adjustment mechanism dynamically adapts the features based on the evolving solution space explored by the fish swarm.

This step ensures that the selected features align with the swarm's collective intelligence, enhancing the discriminatory power of the MSVM.

$$Feature(t+1) = AdjustFeatures\begin{pmatrix} Features(t), \\ Pos_{GlobalBest}(t) \end{pmatrix} \quad (32)$$

The adjustment of features is influenced by the collective behavior of the fish swarm, particularly the global best position. The features are adaptively modified to align with the characteristics of the global best solution. This ensures that the selected features contribute effectively to the optimization goal defined by MSVM, considering the evolving preferences of the fish swarm.

$$\Delta Features(t) = SwarmInfluence(Pos_{GlobalBest}(t)) \quad (33)$$

The feature adjustment process involves assigning adaptive weights to the features based on their relevance to the evolving optimization landscape. The weights dynamically change to emphasize features that contribute significantly to the MSVM's classification accuracy. The adaptive feature weights enhance the discriminative capability of the selected features, aligning them with the swarm's evolving preferences.

$$W_{Features}(t+1) = AdaptWeights(W_{Features}(t), \Delta Features(t)) \quad (34)$$

The adjusted features and their corresponding adaptive weights are integrated into the MSVM framework for classification. This step ensures that the optimized feature set, influenced by the collective intelligence of the fish swarm, is utilized in the SVM decision-making process. The integrated feature set enhances the model's ability to discriminate between classes, improving classification accuracy.

$$MSVM_{Input}(t+1) = IntegrateFeatures(Features(t+1), W_{Features}(t+1)) \quad (35)$$

The feature adjustment process is iterative, reflecting the dynamic nature of the optimization landscape explored by the fish swarm. As the swarm progresses through multiple iterations, the features continuously adapt to align with the evolving preferences of the global best solution.

This iterative refinement ensures that the feature set remains relevant and effective in optimizing MSVM's performance throughout the optimization process.

$$Features(t+i) = AdjustFeatures\left(Features(t+i-1), Pos_{GlobalBest}(t+i-1)\right) \quad (36)$$

The feature adjustment by the CFO in CFO-MSVM represents a critical step in leveraging the collective intelligence of the fish swarm to optimize the feature set for enhanced MSVM performance.

The adaptive modification of features, influenced by the swarm's dynamics, contributes to the overall success of the optimization framework in achieving improved classification accuracy.

3.8. Optimal Kernel Parameter Determination

The CFO-MSVM algorithm incorporates a step for determining the optimal kernel parameters, which is crucial for achieving robust classification performance. The optimization process dynamically adjusts the kernel parameters, avoiding manual tuning and ensuring adaptability to the evolving solution space explored by the fish swarm.

$$\Theta_{Optimal}(t+1) = OptimizeKernelParameters(\Theta(t), Pos_{GlobalBest}(t)) \quad (37)$$

The swarm's dynamics guide the adaptation of kernel parameters to align with the evolving optimization landscape. This ensures that the selected kernel configuration contributes effectively to the optimization goal defined by MSVM.

$$\Delta\Theta(t) = SwarmInfluence(Pos_{GlobalBest}(t)) \quad (38)$$

The adaptive integration of the kernel function involves incorporating the optimized kernel parameters into the MSVM framework. The dynamically adjusted kernel parameters enhance the model's ability to capture complex patterns within the data, improving the discriminative power of the MSVM. The integration ensures that the kernel function aligns with the evolving preferences of the fish swarm.

$$K_{Adaptive}(t+1) = IntegrateKernelFunction\left(K(t), \Theta_{Optimal}(t+1)\right) \quad (39)$$

The kernelized MSVM input is the result of combining the feature-adjusted input and the adaptive kernel function. This integrated input, influenced by both the collective intelligence of the fish swarm and the optimized kernel parameters, forms the foundation for robust and accurate classification. The kernelized MSVM input reflects the dynamic nature of the optimization process and ensures the model is equipped to handle intricate data patterns.

$$MSVM_{Kernelized}(t+1) = IntegrateKernelizedInput\left(MSVM_{Input}(t+1), K_{Adaptive}(t+1)\right) \quad (40)$$

Adjusting the kernel parameters is an iterative process, similar to the refinement of the iterative feature. As the fish swarm progresses through multiple iterations, the kernel parameters continuously adapt to align with the evolving preferences of the global best solution. This iterative refinement ensures that the kernelized MSVM input remains relevant and effective in optimizing classification accuracy throughout the optimization process.

$$\Theta_{Optimal}(t+i) = OptimizeKernelParameters\left(\Theta_{Optimal}(t+i-1), Pos_{GlobalBest}(t+i-1)\right) \quad (41)$$

Integrating the kernel function in CFO-MSVM is crucial in enhancing the model's ability to capture complex data patterns. The dynamic adjustment of kernel parameters,

influenced by the collective intelligence of the fish swarm, ensures adaptability and robustness in achieving optimal classification performance.

3.9. Objective Function Definition

The optimization objective in CFO-MSVM is grounded in the well-established concept of SVM's objective function, seeking to minimize the classification error while maximizing the margin between different classes. The objective function encapsulates the essence of the algorithm's goal, defining a measure that the CFO-MSVM endeavours to optimize.

$$J(w,b) = \frac{1}{2}\|W\|^2 + C\sum_{i=1}^{N}\xi_i \quad (42)$$

Where w represents the weight vector, b is the bias term, ξ_i denotes the slack variables, and C is the regularization parameter. The CFO-MSVM seamlessly integrates fish swarm optimization into the objective function, enhancing its adaptability and convergence efficiency. The swarm's influence is embedded in the weight vector and bias term adjustments, ensuring a collective effort to explore the solution space and refine the classification model.

$$J_{CFO}(w,b,S) = J(w,b) + \alpha.SwarmEffect(\boldsymbol{S}) \quad (43)$$

Where α regulates the influence of the swarm, and $SwarmEffect(S)$ captures the collective impact of the fish swarm on the optimization process.

The optimization objective involves continuously adjusting the bias term and weight vector under the influence of the fish swarm. The swarm dynamically guides the optimization process, ensuring that the SVM's decision boundary aligns with the evolving optimal solution.

$$b(t+1) = b(t) + \beta.SwarmEffect(S)$$
$$w(t+1) = w(t) + \gamma.SwarmEffect(S) \quad (44)$$

Where β and γ control the extent of bias and weight adjustments, respectively. CFO-MSVM introduces a mechanism to adaptively update the regularization parameter C based on the collective behavior of the fish swarm. This ensures that the optimization process considers the varying importance of regularization for different regions of the solution space.

$$C(t+1) = C(t) + \delta.SwarmEffect(S) \quad (45)$$

Where δ governs the rate of regularization parameter adjustment. The CFO-MSVM refines the classification margin, a critical aspect of SVM, through the collaborative efforts of the fish swarm. The swarm's influence guides the margin adjustment, making the algorithm handle difficult classification scenarios.

$$Margin(t+1) = Margin(t) + \epsilon.SwarmEffect(S) \quad (46)$$

Where ϵ controls the rate of margin adjustment based on the swarm's impact.

3.10. Dual Problem Formulation

The CFO-MSVM's tenth step delves into solving the dual problem, a pivotal phase in the optimization process. The dual problem arises from the Lagrangian dualization of the primal SVM problem, providing an alternative perspective that facilitates efficient optimization.

$$L_D(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j X_i^T X_j \quad (47)$$

The dual problem seeks to maximize this Lagrangian function, which is subject to the constraints imposed by the SVM's primal problem. The CFO-MSVM injects the swarm's influence into the Lagrangian function, creating a dynamic and adaptive optimization landscape. The collective behaviour of the fish swarm contributes to the Lagrangian function, aligning the optimization process with the swarm's exploration and exploitation tendencies.

$$L_{D_{CFO}}(\alpha,S) = L_D(\alpha) + \xi.SwarmEffect(S) \quad (48)$$

Where ζ controls the impact of the swarm on the Lagrangian function.

The optimization process involves ascending the gradient of the swarm-influenced Lagrangian function to iteratively approach the optimal dual solution. The fish swarm guides this ascent, ensuring that the optimization aligns with the collective intelligence of the swarm.

$$\nabla L_{D_{CFO}}(\alpha,S) = \nabla L_D(\alpha) + \eta.SwarmEffect(S) \quad (49)$$

Where η dictates the rate of gradient ascent influenced by the swarm.

The CFO-MSVM dynamically updates the dual solution, adapting to the evolving Lagrangian landscape under the influence of the fish swarm. The swarm's collective intelligence takes the dual solution, optimizing it for enhanced classification accuracy.

$$\alpha(t+1) = \alpha(t) + \mu.SwarmEffect(s) \quad (50)$$

Where μ governs the rate of dual solution updates based on the swarm's impact.

The CFO-MSVM introduces swarm-informed convergence criteria to determine when the optimization achieves the desired convergence. The collective behaviour of the fish swarm influences the convergence assessment, aligning it with the algorithm's overarching goals.

$$Convergence(t) = Convergence(t-1) + \xi.SwarmEffect(S) \quad (51)$$

The parameter ξ in Equation (51) controls the convergence assessment influenced by the swarm. The culmination of the tenth step results in the final dual solution,

dynamically shaped by the collaborative efforts of the fish swarm. The adaptive optimization process ensures that the dual solution aligns with the swarm's exploration and exploitation tendencies, optimizing it for robust classification performance.

$$Final\ Dual\ Solution = \alpha(T) \quad (52)$$

Where T represents the iteration at which the convergence criteria are met.

3.11. Modification of Decision Function

The eleventh step in the CFO-MSVM intricately addresses modifying the decision function. This modification is pivotal in adapting the SVM's decision boundary based on the optimized dual solution and the collaborative influence of the fish swarm.

$$f(X) = \sum_{i=1}^{N} \alpha_i y_i X_i^T x + b \quad (53)$$

Equation (53) integrates the optimized dual solution α, class labels y_i, and input feature vectors X_i. The CFO-MSVM introduces a novel dimension by integrating the influence of the fish swarm into the decision function. This swarm-driven modification ensures that the decision function aligns with the collective intelligence of the swarm, optimizing it for robust classification in complex scenarios.

$$f_{CFO}(x, S) = f(x) + \gamma . SwarmEffect(S) \quad (54)$$

Where γ governs the impact of the swarm on the decision function.

The decision function's modification results in an adaptive decision boundary that dynamically responds to the evolving optimization landscape shaped by the fish swarm. The adaptive nature of the decision boundary enhances the CFO-MSVM's resilience in handling diverse and challenging classification scenarios.

$$Decision\ Boundary_{CFO}(f_{CFO}) = \{X | f_{CFO}(x, S) = 0\} \quad (55)$$

The modification extends to the SVM's margin, with the CFO-MSVM incorporating the swarm's guidance to adapt the margin based on the optimized decision function. This margin improves the ability of the algorithm to generalize and classify instances with improved robustness.

$$Margin_{CFO} = \frac{2}{\|\alpha\|} \quad (56)$$

The CFO-MSVM's margin adaptation is dynamically influenced by the optimized dual solution and the swarm's collaborative effect. The CFO-MSVM ensures real-time updates to the decision function as the fish swarm collectively influences the optimization process. This real-time adaptability aligns the decision function with the evolving optimization landscape, promoting accurate and dynamic classification. The parameter δ controls the rate of real-time decision function updates influenced by the swarm.

$$f_{CFO}(x, S, t+1) = f_{CFO}(x, S, t) + \delta . Swarmeffect(S) \quad (57)$$

The CFO-MSVM's modified decision function culminates in a classification decision rule that dynamically classifies input instances based on the optimized decision boundary. The adaptive nature of the decision rule ensures accurate and context-aware classification, which is essential for handling complex scenarios.

$$Classify(x) = \begin{cases} +1, & if\ f_{CFO}(x, S) > 0 \\ -1 & otherwise \end{cases} \quad (58)$$

The classification decision rule integrates the optimized decision function and the swarm's influence, ensuring robust classification outcomes.

3.12. Optimal Kernel Parameters

In the twelfth step of CFO-MSVM, paramount attention is directed towards fine-tuning kernel parameters, a crucial aspect for achieving optimal classification performance. The selection of kernel parameters mostly influences the CFO-MSVM's ability to capture complex relationships within the data.

$$K(x_i, x_j;\ \Theta) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} \quad (59)$$

The Radial Basis Function (RBF) kernel serves as a pivotal component, and its parameter σ undergoes meticulous tuning to strike a balance between model complexity and generalization.

The CFO-MSVM introduces a novel dimension to parameter tuning by incorporating the collaborative guidance of the fish swarm. The swarm dynamically influences the optimization landscape, ensuring that the selected kernel parameters align with the collective intelligence of the swarm.

$$\sigma_{CFO} = \sigma + \beta . SwarmEffect(S) \quad (60)$$

Where β controls the degree to which the fish swarm impacts the adjustment of the kernel parameter σ.

The tuning process extends to the learning rate, a critical parameter governing the convergence of the optimization algorithm. The CFO-MSVM introduces an adaptive learning rate mechanism influenced by the optimization landscape and the swarm's collective impact.

$$\sigma_{CFO} = \eta + \gamma . SwarmEffect(S) \quad (61)$$

Where γ regulates the extent to which the fish swarm guides the adjustment of the learning rate η. The regularization parameter, essential for controlling overfitting, undergoes meticulous adjustment to balance fitting the training data and maintaining model simplicity. The CFO-MSVM integrates the

influence of the fish swarm to adapt the regularization parameter dynamically.

$$C_{CFO} = C + \alpha . SwarmEffect(S) \quad (62)$$

Where α governs the impact of the fish swarm on the adjustment of the regularization parameter C.

A distinctive feature of CFO-MSVM's parameter tuning is its real-time adaptability. As the fish swarm collectively influences the optimization landscape, the parameters are dynamically updated in real-time to align with the evolving context.

$$\Theta_{CFO}(t+1) = \Theta_{CFO}(t) + \delta . SwarmEffect(S) \quad (63)$$

Where δ controls the rate of real-time updates to the kernel parameters influenced by the fish swarm.

The culmination of the parameter tuning process in CFO-MSVM ensures enhanced generalization and adaptability. The adjusted kernel parameters, learning rate, and regularization parameter collectively contribute to a model that adeptly navigates the intricacies of the data landscape, resulting in improved classification accuracy and robustness.

$$\Theta_{CFO} = \{\sigma_{CFO}, \eta_{CFO}, C_{CFO}\} \quad (64)$$

The optimized parameter set Θ_{CFO} encapsulates the dynamically tuned values, reflective of the collaborative influence of the fish swarm in achieving an optimally performing CFO-MSVM.

3.13. Kernel Matrix Computation

In the thirteenth step of CFO-MSVM, the pivotal phase of model training unfolds with the computation of the kernel matrix. This matrix encapsulates the pairwise similarities between data points, facilitating the transformation of the data points into a solution space.

$$K = \begin{bmatrix} K(X_1, X_1) \dots K(X_1, X_n) \\ \vdots \ddots \vdots \\ K(X_n, X_1) \dots K(X_n, X_n) \end{bmatrix} \quad (65)$$

Where K is basic for the subsequent training phases, capturing the intricate relationships embedded within the input data.

CFO-MSVM's training proceeds by formulating the dual problem, an essential step in unleashing the power of support vector machines. The dual problem seeks to optimize the Lagrangian dual function, introducing a set of Lagrange multipliers (α) associated with the training samples.

$$min_{\alpha} \left(\frac{1}{2} \alpha^T H \alpha - 1^T \alpha \right) \quad (66)$$

Where matrix H is defined as $H = Y \odot (K + \gamma I) \odot Y$, incorporating the kernel matrix, target labels Y, and the regularization parameter γ. CFO-MSVM leverages specialized optimization solvers to tackle the intricacies of the dual problem. These solvers employ iterative techniques, updating the Lagrange multipliers (α) until convergence is achieved.

$$\nabla_{\alpha} = H\alpha - 1 \quad (67)$$

Where gradient ∇_{α} guides the optimization process, steering towards the optimal Lagrange multipliers that define the vector points in space.

With the optimized Lagrange multipliers, CFO-MSVM identifies the support vectors and crucial data points that significantly influence the decision boundary. Support vectors have non-zero Lagrange multipliers and are pivotal in shaping the classification model-based data distribution.

$$\alpha_i > 0 \Rightarrow x_i \text{ is a support vector} \quad (68)$$

The identification process ensures that the training focuses on the data points, improving the general model capacity. The culmination of the training of the model in phase involves the calculation of the weight vector (w) and bias term (b). These components collectively define the decision function of CFO-MSVM, mapping input data to class labels.

$$b = \frac{1}{|SV|} \sum_{i \in SV} \left(Y_i - \sum_{j=1}^{n} \alpha_k Y_j K(x_j, x_i) \right) \quad (69)$$

Where the weight vector w captures the weighted contributions of support vectors, while the bias term b ensures the appropriate translation of the decision boundary.

The trained CFO-MSVM model unleashes its decision function to classify new, unseen data. The decision function calculates the confidence scores, allowing the model to assign class labels confidently based on the input features.

$$f(x) = sign(w^T x + b) \quad (70)$$

The CFO-MSVM's decision function exhibits the culmination of the training process, offering a robust and well-informed mechanism for classification tasks. The final step of CFO-MSVM encompasses the application of the trained model's decision function to assess its performance on unseen data. This phase involves leveraging the learned parameters—weight vector (w) and bias term (b)—to classify instances based on their feature representations.

$$f(x) = sign(w^T . \emptyset(X_{mod}) + b) \quad (71)$$

The decision function yields predictions for each data point, indicating the assigned class labels. CFO-MSVM's efficacy is quantified through evaluation metrics that gauge its classification performance. These metrics include precision, recall, F1 score, and accuracy, providing a comprehensive assessment of the model's strengths and areas for improvement.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (72)$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negative} \quad (73)$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (74)$$

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions} \quad (75)$$

To delve deeper into the model's performance, CFO-MSVM constructs a confusion matrix to find the TP, TN, FP, and FN distribution of the predictions.

$$Confusion\ Matrix = \begin{bmatrix} True\ Positives & False\ Positives \\ False\ Negatives & True\ Negatives \end{bmatrix} \quad (76)$$

The confusion matrix serves as the foundation for various performance metrics and aids in identifying specific areas of model misclassification. Beyond binary metrics, CFO-MSVM embraces Receiver Operating Characteristic (ROC) curve analysis for a nuanced evaluation. The ROC curve illustrates the trade-off between true and false positive rates across different decision thresholds.

$$True\ Positive\ Rate(Sensitivity) = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (77)$$

$$False\ Positive\ Rate(1 - Specificity) = \frac{False\ Positives}{False\ Positives + True\ Negatives} \quad (78)$$

ROC analysis provides a holistic view of the model's discriminatory capacity and assists in selecting an optimal decision threshold. Complementing ROC analysis, CFO-MSVM quantifies its discriminative prowess by calculating the Area Under the Curve (AUC). AUC represents the probability that the model will rank a randomly chosen positive instance higher than a randomly chosen negative one.

$$AUC = \int_0^1 True\ Positive\ Rate\ d(False\ Positive\ Rate) \quad (79)$$

The AUC values closer to 1 signify superior discriminatory performance, while values around 0.5 indicate chance-level performance.

The CFO algorithm is very effective in the selection of the starting values and the hyper-parameters. The CFO is very strong and powerful with a simple implementation strategy. The SVM for predicting the defect class lacks strong global optimization, has low convergence speed, and has less optimization precision with reference to the prediction. Modifying the SVM parameters with improved CFO leverages CFO-MSVM classifier fine-tuning and addressing the optimization problem.

Algorithm 1. CFO-MSVM

Input:

- Training dataset $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots (x_n, y_n)\}$ where x_i represents the feature vector and y_i denotes the corresponding class label.
- Parameters:
  - ✓ C (Regularization parameter)
  - ✓ ϵ (Tolerance for convergence)
  - ✓ Maximum number of iterations T

Output:

- CFO-Modified SVM classifier with optimized parameters.

Procedure:

Step 1: Initialization

- Initialize the SVM weight vector w and bias term b to zeros.
- Randomly initialize the parameters of the clownfish optimization algorithm.
- Set the iteration counter t=0.

Step 2: Feature Modification

- Normalize the feature vectors to ensure uniform scaling.
- Perform feature selection or transformation if necessary.

Step 3: CFO Swarm Initialization

- Initialize the clownfish swarm with random positions and velocities.
- Set the personal best position and fitness value for each clownfish.
- Identify the global best position among all clownfish.

Step 4: Objective Function Evaluation

- Evaluate the objective function for each clownfish position using the SVM objective.
- Update the personal best position and fitness value for each clownfish if necessary.
- Update the best fish position based on the clownfish with the best fitness value.

Step 5: Global Best Initialization

- Update the global best position obtained from the clownfish optimization algorithm.
- Extract the SVM parameters (weight vector and bias term) from the global best position.

Step 6: Swarm Movement

- Update the clownfish positions and velocities using the clownfish optimization algorithm.
- Apply velocity limits and boundary constraints if necessary.

Step 7: Feature Adjustment by CFO
- Adjust the features of the training dataset based on the clownfish positions.
- Modify the feature space according to the clownfish swarm's influence.

Step 8: Kernel Function Integration
- Incorporate the modified feature space into the SVM kernel function.
- Compute the kernel matrix based on the adjusted feature vectors.

Step 9: Optimization Objective
- Formulate the optimization objective using the SVM loss function and regularization term.
- Apply optimization techniques to minimize the objective function.

Step 10: Dual Problem Solution
- Solve the dual problem of the SVM optimization using the kernel matrix and label vector.
- Obtain the optimal Lagrange multipliers (alphas) corresponding to support vectors.

Step 11: Decision Function Modification
- Compute the decision function based on the optimized SVM parameters.
- Adjust the decision threshold if necessary.

Step 12: Parameter Tuning
- Fine-tune the SVM C hyperparameters using cross-validation or grid search.
- Optimize the clownfish optimization parameters based on the model performance.

Step 13: Training
- Train the SVM with the modified feature space and optimized parameters.
- Iterate until convergence or the maximum number of iterations is reached.

Step 14: Model Evaluation
- Evaluate the trained SVM model with accuracy, precision, recall and F1 score.
- Assess the model's generalization ability on unseen data using cross-validation or a separate test dataset.

4. Dataset

Software Defect prediction relies on the dataset used publicly. NASA defect repository is no longer used in SDP. AEEEM (Appraisal-Based Estimation of Effort) repository is considered for training the SDP models and proposed by M. D'Ambros et al. JDT is taken to investigate the new proposed model.

The dataset consists of 17 code metrics from the software,17 entropy-of-source-code metrics, 17 churn-of-source-code metrics, 5 entropy-of-change metrics and other related metrics, with a total of 61 together. Table 1 describes the dataset.

Table 1. Dataset Representation

Dataset Used	Java Developers Toolkit
No of Features	61
Total Samples	997
Defect Artifact	206
Non-Defect Artifact	791

Table 2. Parameters used in the CFO-MSVM Algorithm

Setup Parameters	Units Used
Initial Swarm Size (S)	997
Number of Dimensions (D)	2
Maximum Number of Iterations (t_{max})	100
Step Size (StepSize)	0.05
Visual Range (VisualRange)	0.2
Individual Step Size (IndividualStepSize)	0.05
Step Size Reduction Factor (StepSizeReduction)	0.50
Stopping Criteria	100
Initial Population	Random

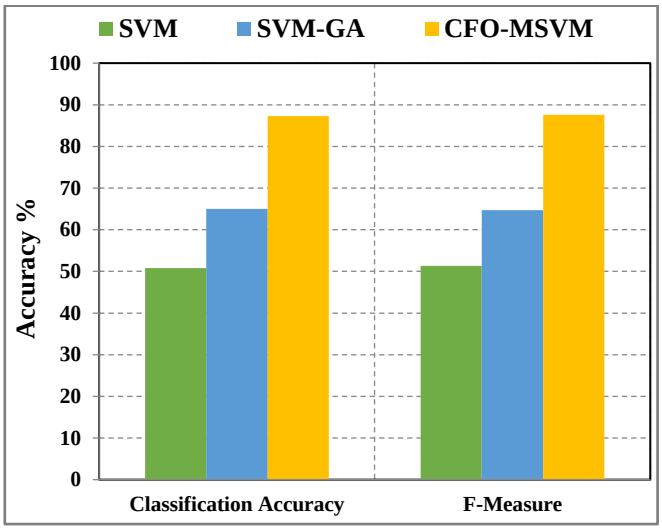

Fig. 1 Accuracy and F Measure for CFO-MSVM

5. Results and Discussion

The classification accuracy of the proposed model CFO-MSVM is analyzed by incorporating the software metric of software with its log file. Code Metrics contains LoC, Halstead Complexity, and McCabe Complexity for given modules of the software. The parameters for evaluation were set up for optimization, as shown in Table 2.

5.1. Classification Accuracy and F-Measure Analysis

Figure 1 comprehensively compares classification accuracy and F-measure metrics across three distinct classifiers: SVM, SVM-GA, and CFO-MSVM. These metrics serve as crucial benchmarks for evaluating the working and performance of classification models in various current running applications, emphasizing the importance of precision and reliability in classification tasks. Upon close examination

of the provided data in Table 3, the inherent limitations of SVM become apparent. SVM exhibits a modest classification accuracy of 50.833% and an F-measure of 51.315%. One prominent disadvantage of SVM is its difficulty handling large feature sets. As datasets become increasingly complex with a high dimensionality of features, SVM's computational complexity escalates, leading to suboptimal performance and compromised accuracy. This limitation impedes SVM's effectiveness in scenarios where datasets contain many features, highlighting the need for alternative approaches to address scalability challenges. SVM-GA showcases improvement over SVM, with a classification accuracy of 65.003% and an F-measure of 64.728%. SVM-GA grapples with its challenges, particularly in effectively handling constraints. While proficient in optimizing parameters, genetic algorithms may struggle to adhere to limitations imposed by the problem space, resulting in compromised performance and suboptimal solutions. This difficulty in handling constraints hampers SVM-GA's ability to achieve optimal classification accuracy, especially in scenarios where strict constraints dictate the problem domain. This underscores the need for robust optimization strategies to overcome constraint-related challenges. CFO-MSVM emerges as a frontrunner in classification accuracy and F-measure, boasting an impressive accuracy of 87.325% and an F-measure of 87.620%. The distinct advantage of CFO-MSVM lies in its unique capability to facilitate parallel implementations, a feature that significantly enhances scalability and computational efficiency. The super power of parallel processing makes the CFO-MSVM adequately check the solution space and optimize SVM parameters, thereby achieving superior classification performance and outperforming its counterparts.

This parallelization capability accelerates computation and enables CFO-MSVM to handle large-scale datasets easily, making it a promising solution for classification tasks in diverse domains. SVM and SVM-GA encounter limitations, such as difficulty handling large feature sets and constraints. CFO-MSVM leverages its advantage of facilitating parallel implementations to overcome these challenges and achieve remarkable performance. The ability of CFO-MSVM to harness parallel processing capabilities not only enhances scalability but also accelerates computation, enabling it to outperform traditional SVM approaches and emerge as a promising solution for classification tasks across various domains. This underscores the significance of adopting innovative optimization strategies to enhance classifier performance and address the evolving needs of modern data analysis and machine learning applications.

Table 3. Classification Accuracy for CFO-MSVM

Classifiers	Classification Accuracy	F-Measure
SVM	50.833	51.315
SVM-GA	65.003	64.728
CFO-MSVM	87.325	87.620

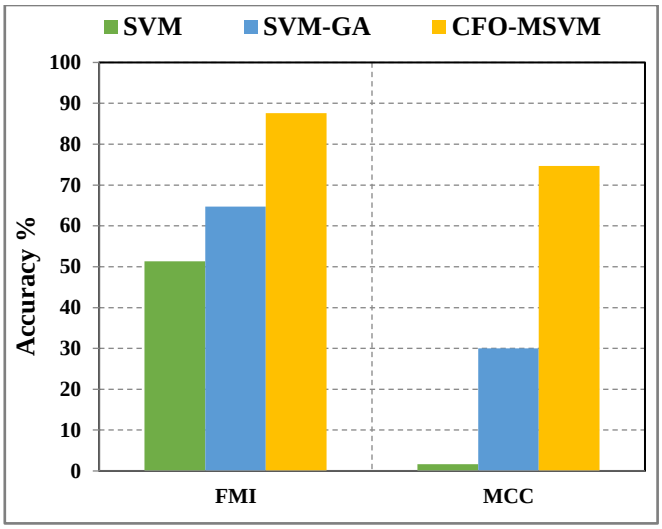

Fig. 2 Statistical Analysis for CFO-MSVM using FMI Index and MCC

Table 4. Statistical Analysis with FMI Index and MCC

Classifiers	FMI	MCC
SVM	51.317	1.663
SVM-GA	64.730	30.007
CFO-MSVM	87.623	74.649

5.2. FMI and MCC Analysis

Figure 2 offers a comprehensive evaluation of the performance of CFO-MSVM against state-of-the-art algorithms using two critical metrics: the Fowlkes-Mallows Index (FMI) and the Matthews Correlation Coefficient (MCC). Any classification model can be basically benchmarked with the specified metrics to check the model accuracy and usage in real time applications. Upon meticulous examination of the data provided in Table 4, it becomes evident that CFO-MSVM emerges as the frontrunner among its counterparts, outperforming both Support Vector Machine (SVM) and Genetic Algorithm-based Support Vector Machine (SVM-GA) in terms of both FMI and MCC scores. With a robust FMI of 87.623 and an impressive MCC of 74.649, CFO-MSVM showcases exceptional classification accuracy and model quality, indicating its effectiveness in various classification tasks.

Among the exemplary performance of CFO-MSVM, it is imperative to delve into the inherent disadvantages of SVM and SVM-GA. Despite its widespread adoption, SVM grapples with the challenge of handling imbalanced data. The intrinsic bias towards majority classes in imbalanced datasets poses a significant obstacle for SVM, as traditional formulations struggle to learn from minority classes, resulting in skewed models effectively and compromised classification accuracy. This limitation overshadows SVM's overall performance, manifested in lower FMI and MCC scores than more adept models. A proper balance between exploration and exploitation becomes a hurdle in the SVM-GA classifier. Genetic algorithms, renowned for their efficacy in parameter

optimization, often struggle to strike an optimal balance between exploring new regions of the solution space and exploiting promising solutions.

This imbalance can impede the optimization process, resulting in premature convergence or insufficient exploration, ultimately diminishing the model's classification prowess and dampening FMI and MCC scores. CFO-MSVM harnesses its unique advantage: hybridization potential by synergistically integrating the Clown Fish Optimization (CFO) algorithm with Support Vector Machines (SVM), CFO-MSVM pioneers a hybrid approach that elevates both exploration and exploitation facets of the optimization process. The CFO algorithm's adeptness in traversing the solution space harmonizes seamlessly with SVM's robust classification capabilities, culminating in a synergy that propels CFO-MSVM to achieve exceptional parameter optimization and model refinement. This hybridization prowess empowers CFO-MSVM to attain unparalleled FMI and MCC scores, surpassing traditional SVM and SVM-GA paradigms and positioning it as a vanguard solution for intricate classification tasks across diverse domains. SVM and SVM-GA grapple with intrinsic limitations, and CFO-MSVM capitalizes on its hybridization prowess to transcend these challenges and spearhead the realm of classification modelling. The fusion of the CFO algorithm's exploratory acumen with SVM's classification finesse underscores the pivotal role of hybrid methodologies in navigating the complexities of real-world classification problems, heralding a new era of precision and reliability in classification modelling.

5.3. TPR and TNR Analysis

Figure 3 provides a nuanced insight into the performance evaluation of three prominent classifiers: SVM, SVM-GA, and CFO-MSVM. The metrics under scrutiny, True Positive Rate (TPR) and True Negative Rate (TNR), are pivotal indicators of a classifier's sensitivity and specificity, crucial for understanding its efficacy in real-world applications. SVM, a powerful tool in classification tasks, often encounters memory-intensive requirements during training. This high demand for computational resources can pose significant challenges, mainly when dealing with large datasets. The extensive memory requirements not only strain hardware resources but also result in prolonged training times, hindering the scalability and efficiency of SVM models. Consequently, this memory-intensive nature may lead to suboptimal performance in TPR and TNR metrics, as SVM struggles to process and learn from vast amounts of data efficiently.

Table 5. Accuracy Measurement using TPR and TNR metrics

Classifiers	True Positive Rate	True Negative Rate
SVM	51.743	49.920
SVM-GA	65.136	64.874
CFO-MSVM	88.389	86.229

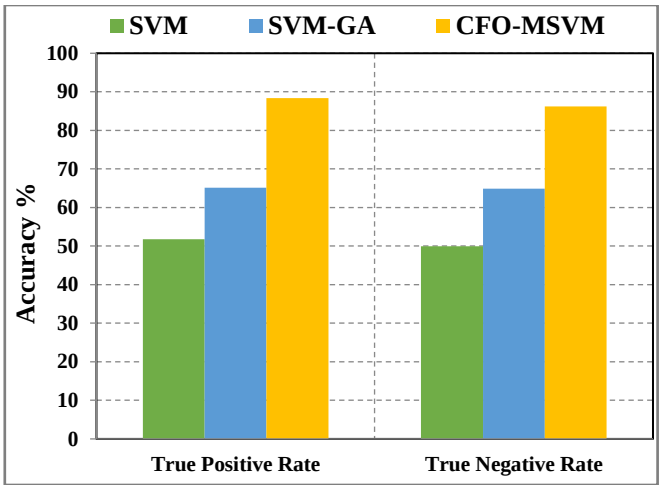

Fig. 3. TPR and TNR metrics

SVM-GA faces its own set of challenges, particularly in the realm of population initialization. Although effective in optimizing parameters, genetic algorithms heavily rely on the initial population of solutions to kickstart the optimization process. However, the quality and diversity of this initial population significantly influence the performance of SVM-GA. Early-term convergence happens due to improper population initialization, making the algorithm work poorly in finding the solution space. Consequently, SVM-GA may exhibit lower TPR and TNR metrics due to inadequate exploration of potential solutions. CFO-MSVM stands out due to its efficient parameter optimization capabilities. By integrating the Clown Fish Optimization algorithm with SVM, CFO-MSVM streamlines the parameter tuning process, facilitating quicker convergence and more effective solution space exploration.

This streamlined optimization approach allows CFO-MSVM to identify and refine optimal parameter configurations more efficiently, improving classification performance and higher TPR and TNR metrics. CFO-MSVM's hybridization potential enables it to leverage the strengths of both optimization techniques, maintaining equivalence of exploration and exploitation to achieve superior classification accuracy. SVM and SVM-GA grapple with memory-intensive training and population initialization challenges, and CFO-MSVM leverages its efficient parameter optimization capabilities to excel in classification tasks. Adopting innovative optimization techniques allows CFO-MSVM to overcome these hurdles and achieve superior performance in TPR and TNR metrics. CFO-MSVM emerges as a promising solution for classification tasks, offering enhanced accuracy and reliability across diverse domains.

6. Conclusion

The integration of Clown Fish Optimized Modified Support Vector Machine (CFO-MSVM) presents a promising approach for software defect prediction. The utilization of the

Modified Clown Fish Optimization (CFO) algorithm on the Support Vector Machine (SVM) effectively fine-tunes and enhances the classification performance. The experiment conducted on the JDT dataset leverages the fact that the CFO-MSVM performs well compared to the conventional SVM classifier. The accuracy and efficiency are increased to 87.32% from 50 %, incorporating the optimization technique on SVM. CFO-MSVM addresses the challenge of hyperparameter tuning much better by combining the power of CFO. The convergence speed is upscaled. CFO-MSVM classifier handles complex datasets, making it suitable for predicting the defects in software. Based on the dataset characteristics and software nature, the efficiency may decrease. In future, some ensemble techniques can be used to address the issue by incorporating some domain-specific knowledge. The potential bugs can be identified early making the CFO-MSVM a reliable and valuable solution for prediction. CFO-MSVM is a valuable tool for improving software quality and reducing maintenance efforts in software engineering practices**.**

References

[1] Emad Shihab et al., "Is Lines of Code a Good Measure of Effort in Effort-Aware Models?," *Information and Software Technology*, vol. 55, no. 11, pp. 1981-1993, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[2] Harold Valdivia-Garcia, Emad Shihab, and Meiyappan Nagappan, "Characterizing and Predicting Blocking Bugs in Open Source Projects," *Journal of Systems and Software*, vol. 143, pp. 44-58, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[3] Cheng Zhou et al., "Leveraging Multi-Level Embeddings for Knowledge-Aware Bug Report Reformulation," *Journal of Systems and Software*, vol. 198, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[4] R. Martí et al., "Providing Early Resource Allocation During Emergencies: The Mobile Triage Tag," *Journal of Network and Computer Applications*, vol. 32, no. 6, pp. 1167-1182, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[5] Bader Alkhazi et al., "Learning to Rank Developers for Bug Report Assignment," *Applied Soft Computing*, vol. 95, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[6] Anjali Goyal, and Neetu Sardana, "Performance Assessment of Bug Fixing Process in Open Source Repositories," *Procedia Computer Science*, vol. 167, pp. 2070-2079, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[7] Fiorella Zampetti et al., "Automating Orthogonal Defect Classification Using Machine Learning Algorithms," *Future Generation Computer Systems*, vol. 102, pp. 932-947, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[8] Fábio Lopes et al., "An Empirical Characterization of Software Bugs in Open-Source Cyber–Physical Systems," *Journal of Systems and Software*, vol. 192, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[9] Thazin Win Win Aung et al., "Multi-Triage: A Multi-Task Learning Framework for Bug Triage," *Journal of Systems and Software*, vol. 184, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[10] Ashima Kukkar et al., "ProRE: An ACO- based Programmer Recommendation Model to Precisely Manage Software Bugs," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 1, pp. 483-498, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[11] Chengbin Pang et al., "Generation-Based Fuzzing? Don't Build a New Generator, Reuse!," *Computers & Security*, vol. 129, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[12] Luiz Alberto Ferreira Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes, "Bug Report Severity Level Prediction in Open Source Software: A Survey and Research Opportunities," *Information and Software Technology*, vol. 115, pp. 58-78, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[13] Heetae Cho, Seonah Lee, and Sungwon Kang, "Classifying Issue Reports According to Feature Descriptions in a User Manual Based on a Deep Learning Model," *Information and Software Technology*, vol. 142, pp. 1-13, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[14] Shirin Akbarinasaji, Bora Caglayan, and Ayse Bener, "Predicting Bug-Fixing Time: A Replication Study Using an Open Source Software Project," *Journal of Systems and Software*, vol. 136, pp. 173-186, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[15] Ashima Kukkar et al., "Bug Severity Classification in Software Using Ant Colony Optimization Based Feature Weighting Technique," *Expert Systems with Applications*, vol. 230, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[16] Ashima Kukkar, Rajni Mohana, and Yugal Kumar, "Does Bug Report Summarization Help in Enhancing the Accuracy of Bug Severity Classification?," *Procedia Computer Science*, vol. 167, pp. 1345-1353, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[17] Thomas Hirsch, and Birgit Hofer, "A Systematic Literature Review on Benchmarks for Evaluating Debugging Approaches," *Journal of Systems and Software*, vol. 192, pp. 1-17, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[18] Liangfu Ge et al., "An Improved System for Long-Term Monitoring of Full-Bridge Traffic Load Distribution on Long-Span Bridges," *Structures*, vol. 54, pp. 1076-1089, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[19] Asmita Yadav et al., "Ranking of Software Developers Based on Expertise Score for Bug Triaging," *Information and Software Technology*, vol. 112, pp. 1-17, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[20] Yguaratã Cerqueira Cavalcanti et al., "Towards Semi-Automated Assignment of Software Change Requests," *Journal of Systems and Software*, vol. 115, pp. 82-101, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[21] Ashima Kukkar, and Rajni Mohana, "A Supervised Bug Report Classification with Incorporate and Textual field Knowledge," *Procedia Computer Science*, vol. 132, pp. 352-361, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[22] Xin Xia et al., "Elblocker: Predicting Blocking Bugs with Ensemble Imbalance Learning," *Information and Software Technology*, vol. 61, pp. 93-106, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[23] D. Jayaraj et al., "AFSORP: Adaptive Fish Swarm Optimization-Based Routing Protocol for Mobility Enabled Wireless Sensor Network," *International Journal of Computer Networks and Applications*, vol. 10, no. 1, pp. 119-129, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[24] R. Vadivel, and Ramkumar Jaganathan, "QoS-Enabled Improved Cuckoo Search-Inspired Protocol (ICSIP) for IoT-Based Healthcare Applications," *Incorporating the Internet of Things in Healthcare Applications and Wearable Devices*, pp. 109-121, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[25] Lingaraj Mani, Senthilkumar Arumugam, and Ramkumar Jaganathan, "Performance Enhancement of Wireless Sensor Network Using Feisty Particle Swarm Optimization Protocol," *Proceedings of the 4th International Conference on Information Management & Machine Intelligence*, Jaipur, India, pp. 1-5, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[26] J. Ramkumar, and R. Vadivel, "Whale Optimization Routing Protocol for Minimizing Energy Consumption in Cognitive Radio Wireless Sensor Network," *International Journal of Computer Networks and Applications*, vol. 8, no. 4, pp. 455-464, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[27] J. Ramkumar, and R. Vadivel, "Multi-Adaptive Routing Protocol for Internet of Things based Ad-hoc Networks," *Wireless Personal Communications*, vol. 120, no. 2, pp. 887-909, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[28] S.P. Geetha et al., "Energy Efficient Routing in Quantum Flying Ad Hoc Network (Q-Fanet) Using Mamdani Fuzzy Inference Enhanced Dijkstra's Algorithm (MFI-EDA)," *Journal of Theoretical and Applied Information Technology*, vol. 102, no. 9, pp. 3708-3724, 2024. [Google Scholar] [Publisher Link]

[29] Ramkumar Jaganathan, and Ramasamy Vadivel, "Intelligent Fish Swarm Inspired Protocol (IFSIP) for Dynamic Ideal Routing in Cognitive Radio Ad-Hoc Networks," *International Journal of Computing and Digital Systems*, vol. 10, no. 1, pp. 1063-1074, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[30] M.P. Swapna, and J. Ramkumar, "Multiple Memory Image Instances Stratagem to Detect Fileless Malware," *Advancements in Smart Computing and Information Security*, pp. 131-140, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[31] J. Ramkumar et al., "IoT-Based Kalman Filtering and Particle Swarm Optimization for Detecting Skin Lesion," *Soft Computing Applications in Modern Power and Energy Systems*, pp. 17-27, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[32] Nitish Kumar Ojha, Archana Pandita, and J. Ramkumar, "Cyber Security Challenges and Dark Side of AI: Review and Current Status," *Demystifying the Dark Side of AI in Business*, pp. 117-137, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[33] Khushbakht Ali Qamar, Emre Sülün, and Eray Tüzün, "Taxonomy of Bug Tracking Process Smells: Perceptions of Practitioners and an Empirical Analysis," *Information and Software Technology*, vol. 150, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[34] Jyoti Prakash Meher, Sourav Biswas, and Rajib Mall, "Deep Learning-Based Software Bug Classification," *Information and Software Technology*, vol. 166, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[35] Behzad Soleimani Neysiani, Seyed Morteza Babamir, and Masayoshi Aritsugi, "Efficient Feature Extraction Model for Validation Performance Improvement of Duplicate Bug Report Detection in Software Bug Triage Systems," *Information and Software Technology*, vol. 126, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[36] S.K.B. Sangeetha et al., "An Enhanced Multimodal Fusion Deep Learning Neural Network for Lung Cancer Classification," *Systems and Soft Computing*, vol. 6, pp. 1-10, 2024. [CrossRef] [Google Scholar] [Publisher Link]