mathematics

MDPI

Article

Feature Selection Using Golden Jackal Optimization for Software Fault Prediction

Himansu Das [1,*], Sanjay Prajapati [1], Mahendra Kumar Gourisaria [1], Radha Mohan Pattanayak [2], Abdalla Alameen [3] and Manjur Kolhar [4]

[1] School of Computer Engineering, KIIT Deemed to be University, Bhubaneswar 751024, Odisha, India; mkgourisariafcs@kiit.ac.in (M.K.G.)
[2] School of Computer Science & Engineering, VIT-AP University, Amaravati 522237, Andhra Pradesh, India
[3] Computer Science Department, Prince Sattam Bin Abdulaziz University, Al-Kharj 16278, Saudi Arabia
[4] Department of Computer Science, College of Arts and Science, Prince Sattam Bin Abdulaziz University, Al-Kharj 16278, Saudi Arabia
* Correspondence: himanshufcs@kiit.ac.in

Abstract: A program's bug, fault, or mistake that results in unintended results is known as a software defect or fault. Software flaws are programming errors due to mistakes in the requirements, architecture, or source code. Finding and fixing bugs as soon as they arise is a crucial goal of software development that can be achieved in various ways. So, selecting a handful of optimal subsets of features from any dataset is a prime approach. Indirectly, the classification performance can be improved through the selection of features. A novel approach to feature selection (FS) has been developed, which incorporates the Golden Jackal Optimization (GJO) algorithm, a meta-heuristic optimization technique that draws on the hunting tactics of golden jackals. Combining this algorithm with four classifiers, namely K-Nearest Neighbor, Decision Tree, Quadrative Discriminant Analysis, and Naive Bayes, will aid in selecting a subset of relevant features from software fault prediction datasets. To evaluate the accuracy of this algorithm, we will compare its performance with other feature selection methods such as FSDE (Differential Evolution), FSPSO (Particle Swarm Optimization), FSGA (Genetic Algorithm), and FSACO (Ant Colony Optimization). The result that we got from FSGJO is great for almost all the cases. For many of the results, FSGJO has given higher classification accuracy. By utilizing the Friedman and Holm tests, to determine statistical significance, the suggested strategy has been verified and found to be superior to prior methods in selecting an optimal set of attributes.

Keywords: software fault prediction; software defect prediction; feature selection; classification algorithms; golden jackal optimization

MSC: 65D18; 65D19; 68M07

Citation: Das, H.; Prajapati, S.; Gourisaria, M.K.; Pattanayak, R.M.; Alameen, A.; Kolhar, M. Feature Selection Using Golden Jackal Optimization for Software Fault Prediction. *Mathematics* **2023**, *11*, 2438. https://doi.org/10.3390/math11112438

Academic Editor: Guy De Tré

Received: 16 April 2023
Revised: 12 May 2023
Accepted: 16 May 2023
Published: 25 May 2023

1. Introduction

Software's flaws can harm its reliability and quality, necessitating more maintenance and an effort to rectify it. While testing results can aid software development teams in detecting bugs, testing complete software modules is costly and time-consuming. The performance of various software development tasks by individuals can lead to the emergence of multiple software bugs over time, ultimately resulting in user dissatisfaction. Therefore, early identification of software flaws is one of the primary research areas of interest. Software fault prediction [1,2] is the process of spotting potential flaws or defects in software before they happen using data analysis and machine learning methods. This can aid developers in effectively identifying and resolving potential problems, producing software that is of higher quality and contains fewer flaws. There are several approaches to predicting software faults, such as statistical and machine learning methods [3,4]. These

techniques involve analyzing data from past software projects to identify patterns and trends that may indicate potential faults. The data used for this analysis may include testing and debugging logs, source code, and other relevant information. Commonly utilized methods for predicting software faults include DT [5,6], SVM [7], Neural Networks [8], LR [9], and many more. These techniques involve analyzing data to identify patterns and trends indicating potential flaws or bugs. In addition to machine learning techniques, code analysis and testing are employed to predict software faults. These methods involve scrutinizing the software code for potential issues and performing various tests to detect and rectify bugs. Predicting software failures is a vital aspect of software development that can enhance the quality and reliability of software. By effectively identifying and resolving potential issues, developers can reduce the probability of bugs and improve the overall user experience.

In the field of machine learning, one of the critical tasks is Feature Selection (FS) [10,11]. The process entails determining the most significant features that can improve the precision of predictive models. Several methods are available for FS, each with advantages and limitations. We can say FS is a crucial stage in software fault prediction that aids in locating the most important predictors of software faults. Feature selection is required when the available dataset is extensive and includes many features or variables, making it challenging to analyze and interpret the findings accurately. The most crucial characteristics should be chosen to simplify the analysis and increase the precision of the software fault prediction model. For FS in software fault detection, various methods are employed. One of the methods used for FS is the filter method [12]. These methods employ statistical properties such as correlation with the target variable or variance to select features. Some examples of filter methods include chi-square [12] and ANOVA [12,13]. Wrapper methods [14], on the other hand, evaluate subsets of features by training and testing a predictive model on each subgroup. Recursive feature elimination [15] and forward/backward selection [16] are examples of wrapper methods. Embedded methods [17,18] combine feature selection with model training. Lasso [19] and ridge regression [20] are two examples of embedded methods used to identify the essential elements for prediction. The ridge penalty reduces the regression coefficient estimate, but not precisely to zero. For this reason, the incapability of ridge regression to perform variable selection has long been a source of criticism. As a result, penalized regression techniques like elastic-net, adaptive elastic-net, and adaptive-lasso are more beneficial for variable selection. Meanwhile, dimensionality reduction methods aim to reduce the number of features while retaining the most relevant information for prediction. Some examples of dimensionality reduction methods include Principal Component Analysis (PCA) [21] and t-SNE (neighbor encoding) [22]. It should be emphasized that the selection of the method depends on the specific problem at hand. Therefore, trying multiple ways and comparing their performance is often beneficial to select the best one for a specific situation. Feature selection plays a crucial role in software fault prediction by identifying the most significant variables or features that influence the likelihood of software faults.

Researchers have used many different types of FS algorithms. Evolutionary-based algorithms [23] and swarm-based algorithms [24] have been used for the feature selection approaches. This study's primary objective is to enhance classification accuracy while minimizing errors. The Genetic Algorithm (GA) [25] utilizes a computational technique that is guided by natural selection and genetic evolution in living organisms. This method involves utilizing a group of potential solutions, using genetic techniques such as selection, crossover, and mutation to create new possible solutions, and assessing their efficacy based on a specified function. Particle Swarm Optimization (PSO) [26,27] emulates the movement of a set of particles exploring a search space with multiple dimensions, where each particle embodies a prospective solution to the problem. Through continuous evaluation of each particle's fitness, the algorithm adjusts their position and velocity by considering individual and group experiences. Its ultimate goal is to find the most optimal solution. DE (Differential Evolution) [28] is an optimization algorithm that operates on a population of

candidate solutions using a set of operators, including mutation, crossover, and selection, to converge toward the optimal solution. It uses a difference vector to create new solutions and iteratively improves them by comparing their fitness with the current population. Ant Colony Optimization (ACO) [29] is a metaheuristic optimization algorithm that simulates the foraging behavior of ants to find the optimal solution in a search space. The algorithm uses pheromone trails deposited by ants to guide the search toward the optimal solution. Any algorithm's performance depends on how the user sets its parameter values and utilizes them to find the answer, resulting in its output.

Generally, feature selection is a common task in machine learning. The goal is to identify the most relevant subset of features (i.e., input variables) that are most informative for a given prediction task. The process of feature selection involves an optimization problem that seeks to identify the ideal set of features that maximizes the performance of the model while minimizing its complexity. The objective of the paper is to apply effective FS methods to uncover a subset of features that produces a precise and easy-to-interpret model. The objective of feature selection is to improve the quality and usability of the machine learning model for real-world applications.

The main contribution here is it presents a new feature selection method called feature selection using Golden Jackal Optimization (FSGJO) [30]. Golden Jackal Optimization (GJO) in FS aids in identifying the appropriate set of features. GJO mimics the hunting behavior of golden jackals, known for their cooperative hunting strategy and ability to adapt to changing environments. The algorithm consists of a population of candidate solutions, called jackals, that move around the search space for the ideal solution. The advantage of GJO is its ability to handle complex, high-dimensional optimization problems with multiple objectives. GJO is less likely to get stuck in local optima, which can be a problem for other optimization algorithms because GJO uses a combination of exploratory and exploitative search strategies, which allows it to escape local optima and continue searching for better solutions. The efficacy of the newly developed algorithm has been evaluated against several other feature selection (FS) algorithms, including FSGA, FSDE, FSPSO, and FSACO. Thus, it will be an effective method for feature selection. The new algorithm FSGJO and other FS methods have been used on classification models such as KNN, DT, NB, and QDA to check which model is giving the best accuracy output. The results of the new algorithm were compared with other FS methods for their significance. Software developers can enhance the precision and efficiency of their software fault prediction models by carefully selecting the most appropriate features.

The arrangement of this paper is as follows: in Section 2, the literature on feature selection algorithms is explored, while Section 3 introduces the GJO algorithm, and Section 4 explains the FSGJO approach. Section 5 details the experimental results and analysis, followed by a statistical analysis in Section 6. Finally, the conclusion is presented in Section 7.

2. Literature Review

Software fault prediction methods involve analyzing software code, metrics, or historical data to detect possible faults that may occur during the development process or after the software has been released. These techniques can help developers proactively identify and fix potential defects before they become significant issues. After software development, software testing [31] is conducted to ensure that the software meets the defined requirements and to detect and resolve any defects or problems that may have been overlooked during the development phase. In summary, software fault prediction is typically performed during development, while software testing is performed after the software has been developed. Both techniques are essential for ensuring high-quality software meets user requirements. In software fault prediction (SFP), classification [32–37] is a commonly used technique to predict whether a particular module or component of software contains a fault or defect.

Sonali and Divya [38] introduced a model, known as the Linear Twin Support Vector Machine (LSTSVM), to predict defective software modules. The model incorporates feature

selection techniques and was evaluated on four datasets—CM1, PC1, KC1, and KC2. The study reported encouraging outcomes on the latter three datasets. Turabieh et al., in 2019 [39], took a dataset from the Promise repository which was iteratively subjected to three wrapper feature selection (FS) algorithms—(BPSO), (BGA), and (BACO)—which were applied iteratively, and received results with an average of 0.8358 over all datasets. Ezgi and Selma [40] proposed a hybrid approach using an artificial bee colony and differential evolution, which helps to select a relevant set of features without reducing accuracy. Ibrahim et al. conducted a study in 2017 [41] where they utilized the BSA for feature selection and Random Forest as a classifier on PC1, PC2, PC3, and PC4, which resulted in practical outcomes. In the study [42], the authors employed PSO as a feature selection method and the bagging technique as a classifier. They used eleven classifiers and nine samples from the NASA repository. Except for SVM, their future work, all classifier performances improved after comparing the findings with their methodology. On some of the well-known NASA datasets, authors in [43] combined the Centroid Bat Approach (CBA-SVM) and Support Vector Machine (SVM) methods, contrasted the outcomes with those of other ways, and discovered that their strategy was producing promising results. The authors in [44] employed the bagging technique with the GA and PSO metaheuristic methods to enhance performance. They discovered that the results of the two algorithms were comparable, but the combination with bagging exhibited superior outcomes. Authors in [45] used four datasets, namely PC1, PC2, PC3, and PC4, and tested with a correlation-based feature selection technique with five classifiers. Finally, they found out that CFS with RF has the best performance. There are many different FS algorithms, such as the electric field algorithm [46], Jaya Algorithm [47], RHSFOS [48], FSBWO [49], and many more, that can be tested on the software fault prediction datasets. The author in [50] has done feature selection using the firefly algorithm with SVM, KNN, and NB classifiers achieving better classification accuracy with FS.

The FS approaches have various parameters that control the coming accuracy. So, tuning all those parameters is necessary, and for every problem, it will be different.

3. Summary of Golden Jackal Optimization Algorithm

The Golden Jackal Optimization (GJO) algorithm is a meta-heuristic optimization technique that draws ideas from the hunting pattern of golden jackals. This algorithm aims to emulate the hunting strategy of these opportunistic predators, known for their adaptability to diverse environments. By doing so, GJO seeks to solve optimization problems efficiently and effectively.

The primary stages of hunting for a golden jackal pair are outlined below:

1. Locating the prey and advancing towards it.
2. Trapping the prey and agitating it.
3. Attacking and capturing the prey.

Like other meta-heuristics, the GJO is a population-based technique that initiates with a randomized distribution of the first solution across the search space, as shown in Equation (1).

$$X_0 = X_{min} + rand * X_{max} - X_{min} \tag{1}$$

where X_{min} is lower bound, and X_{max} is upper bound, and $rand$ is a function whose value range between 0 to 1.

The initial matrix Prey (X_{prey}) is represented in Equation (2) which is generated during initialization, where the top two fittest members are a pair of jackals.

$$X_{prey} = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,q} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,q} \\ \vdots & \vdots & \vdots & \vdots \\ X_{p,1} & X_{p,2} & \cdots & X_{p,q} \end{bmatrix} \tag{2}$$

As shown in Equation (2), it involves p preys and q variables. The position (location) of each prey represents the parameters of a particular solution. As part of the optimization procedure, a fitness function is utilized to assess the appropriateness of each prey. As described in Equation (3), the fitness values of all prey are gathered in a matrix, where the F matrix holds the fitness values of every prey. $X_{p,q}$ represents the value of the p_{th} dimension of the q_{th} prey. The optimization involves p preys, and the objective function is denoted by F. In the hunting patterns of golden jackals, the male jackal is considered the most suitable prey, followed by the female jackal as the second fittest. The positions of the prey are acquired by the jackal pair accordingly.

$$\begin{bmatrix} f\left(X_{1,1}; X_{1,2}; \cdots X_{1,q}\right) \\ f\left(X_{2,1}; X_{2,2}; \cdots X_{2,q}\right) \\ \vdots \\ f\left(X_{P,1}; X_{P,2}; \cdots X_{p,q}\right) \end{bmatrix} \tag{3}$$

Due to their inherent nature, jackals are adept at identifying and pursuing prey, but at times the prey may prove elusive and manage to evade them. As a result, the jackals must resort to exploring alternative prey, and this is referred to as the exploration stage. The male jackal is responsible for leading the hunt, with the female jackal following in pursuit. The updated position of the male jackal is shown in Equations (4) and (5), where i corresponds to the current iteration. The prey's position vector is denoted by X_{prey}. X_{FM} represents the location of the female jackal, and X_M represents the location of the male jackal. The revised position of male jackal is symbolized as X_1, and the revised position of female jackal is symbolized as X_2 with respect to the prey. The energy the prey uses to evade is represented by e and is determined using Equation (6).

$$X_1 = X_M(i) - e\left|X_M(i) - s1 * X_{prey}(i)\right| \tag{4}$$

$$X_2 = X_{FM}(i) - e\left|X_{FM}(i) - s1 * X_{prey}(i)\right| \tag{5}$$

$$e = e_0 * e_1 \tag{6}$$

where e_0 denotes the initial energy and e_1 indicates the decreasing energy of the prey.

$$e_0 = 2 * r - 1 \tag{7}$$

$$e_1 = c_1 * \left(1 - \frac{i}{I}\right) \tag{8}$$

The variable of Equations (7) and (8) includes r as a random integer whose value ranges between (0,1) and c_1 represents a constant value set at 1.5. The I signifies the max number of iterations and current iteration is denoted by i. Additionally, the value of e_1 is gradually reduced in a linear manner from 1.5 to 0 over the course of the iterations.

Equations (4) and (5) involve the calculation of the distance between the jackal and prey, represented as $X(i) - s1 * X_{prey}(i)$. Depending on the evading energy of the prey, this distance is either added or subtracted from the current position of the jackal. The two equations utilize a vector $s1$, which consists of a set of random numbers that adhere to the Levy distribution and signify the Levy movement. To simulate the movement of the prey in a Levy fashion, the equation multiplies the vector $s1$ with the Prey vector, as shown in Equation (9).

$$s1 = 0.05 * LF(x) \tag{9}$$

The levy flight function, denoted by $LF(x)$, is computed in Equations (10) and (11), where v ranges between 0 to 1 and δ is generally set to 1.5.

$$LF(x) = \frac{0.05 * \sigma_u}{v^{\frac{1}{\delta}}} \tag{10}$$

$$\sigma_u = \left[\frac{\Gamma(1+\delta) * \sin\left(\frac{\pi\delta}{2}\right)}{\Gamma\left(\frac{1+\delta}{2}\right) * \delta * 2^{\frac{\delta-1}{2}}} \right]^{\frac{1}{\delta}} \tag{11}$$

Finally, the Equation (12) shows the updated positions of the jackals are obtained by averaging the results of Equations (4) and (5).

$$X(i+1) = \frac{X_1(i) + X_2(i)}{2} \tag{12}$$

In a mathematical model, the cooperative hunting behavior of a male jackal and female jackal is represented in Equations (13) and (14), respectively, where i denotes the current iteration, X_{prey} refers to the position vector of the prey, and $X_M(i)$ refers to the location of the male jackal, and $X_{FM}(i)$ refers to the location of female jackal. $X_1(i)$ represents the revised positions of the male jackal, and $X_2(i)$ represent the revised positions of female jackal with respect to the prey. The position updates of the jackals are determined by Equations (6) and (12), which are utilized to compute the evading energy of the prey. To avoid getting stuck in local optima and encourage exploration, Equations (13) and (14) incorporate the function $s1$. The use of Equation (9) to compute $s1$ is aimed at overcoming any sluggishness towards local optima, especially in the later iterations. This factor is akin to the obstacles that jackals face while pursuing prey in their natural habitat. During the exploitation stage, $s1$ serves the purpose of addressing these obstacles.

$$X_1(i) = X_M(i) - e\left|s1 * X_M(i) - X_{prey}(i)\right| \tag{13}$$

$$X_2(i) = X_{FM}(i) - e\left|s1 * X_{FM}(i) - X_{prey}(i)\right| \tag{14}$$

To sum up, the GJO algorithm starts by creating a random prey population as a potential solution. During each iteration of the algorithm, the jackals work together to anticipate the potential location of their prey. Every individual in the population adjusts the distance between the jackal pairs according to the specified criterion. The parameter e_1 is decreased from 1.5 to 0 over time to balance exploration and exploitation. If e exceeds 1, the golden jackal pairs move farther from the prey. In contrast, if e is less than 1, the teams move closer to the prey to increase the chances of capturing it.

4. Feature Selection Using Golden Jackal Optimization

Feature selection refers to picking a smaller relevant subset of predictor variables from a larger dataset, aiming to enhance the accuracy of machine learning models, decrease computational expenses, and reduce the chances of overfitting. Put differently, it is a method of determining the essential features with the highest impact on the target variable. Naturally, selecting parts for classification is difficult; therefore, FSGJO uses GJO optimization to select a relevant subset of features. Using FSGJO, there is an increase in classification accuracy. Below is an explanation of the various phases of FSGJO.

4.1. Initialization

GJO is a population-based approach, similar to various other metaheuristics; the search space is uniformly explored starting from an initial or first solution. The initial

solution is shown in Equation (15), where X_{min} is lower bound, X_{max} is upper bound, and $rand()$ is a function whose value range between 0 to 1.

$$X_{initial} = X_{min} + rand() \cdot (X_{max} - X_{\min}) \tag{15}$$

Suppose there are p preys and q variable; then, an individual can be represented as shown in Equation (16), where $1 \leq i \leq p$ is the index of each prey. However, the population of the prey is directly represented by a $p \times q$ matrix such that, $X_{prey} = (x_{ij})_{p \times q}$ as shown in Equation (17), where $i = 1, 2, 3, \ldots, p$, $j = 1, 2, 3, \ldots, q$, and a row represents an individual prey, and a column represents a dimension (variable). X_{prey} is the initial matrix of prey generated during initialization, where the top two fittest members are a pair of jackals (male jackal and female jackal, respectively).

$$X_i = x_{i1} + x_{i2} + x_{i3} + \cdots + x_{iq} \tag{16}$$

$$X_{prey} = X_{ij} = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,q} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,q} \\ \vdots & \vdots & \vdots & \vdots \\ X_{p,1} & X_{p,2} & \cdots & X_{p,q} \end{bmatrix} \tag{17}$$

The optimization process involves p preys and q variables. The position of each prey represents the parameters of a particular solution. In order to assess the performance of each candidate solution during the optimization process, a fitness function (also known as objective function) is utilized, and the output values of this function for all solutions are stored in a matrix as shown in Equation (18), where, $i = 1, 2, 3, \ldots, p$, $j = 1, 2, 3, \ldots, q$, and fitness values of each prey are stored in a matrix called F_{ij}, where the notation $X_{p,q}$ refers the value of the p_{th} prey on the q_{th} dimension. The optimization involves p preys, and the objective function is denoted by F_{ij}. The male and female jackals acquire the positions of the fittest and second fittest prey, respectively, and these are known as the male jackal and female jackal prey positions.

$$F_{ij} = \begin{bmatrix} f(X_{1,1}; X_{1,2}; \cdots X_{1,q}) \\ f(X_{2,1}; X_{2,2}; \cdots X_{2,q}) \\ \vdots \\ f(X_{P,1}; X_{P,2}; \cdots X_{p,q}) \end{bmatrix} \tag{18}$$

4.2. Exploration Phase

In GJO, exploration is achieved by simulating the movement of a golden jackal pack searching for food in an unknown territory. Each jackal (solution) moves randomly within a specific range to explore the search space. This behavior helps prevent the algorithm from being trapped in local optima and facilitates discovering new solutions. Although, occasionally, the prey cannot be easily grabbed and manages to escape, it is in the nature of the jackal to be able to perceive and track it. Thus, if the prey is not easily caught, the jackals enter the exploration stage, searching for other potential targets. During hunting, the female jackal follows behind while the male jackal takes the lead. The updated position of male jackal is shown in Equations (19) and (20), where variable X_{prey} refers to the location vector of the prey, X_M is the location of the male jackal, and X_{FM} is the location of the female jackal. Variable i represents the current iteration. X_a is the revised positions of the male jackal (X_M), and X_b indicates the revised positions of the female jackal (X_{FM}) in relation to prey. The calculation of the prey's evading energy, E_p, involves Equation (21), wherein the initial energy of the prey can be represented by E_{p0}, while E_{p1} signifies the reduction of its energy.

$$X_a = X_M(i) - E_p \left| X_M(i) - s1 * X_{prey}(i) \right| \tag{19}$$

$$X_b = X_{FM}(i) - E_p \left| X_{FM}(i) - s1 * X_{prey}(i) \right| \tag{20}$$

$$E_p = E_{p0} * E_{p1} \tag{21}$$

E_{p0} is calculated using Equation (22), and E_{p1} is calculated using Equation (23), where r, that is a random number between 0 and 1, as well as a constant value denoted as c_1 that is equal to 1.5. Additionally, the maximum number of iterations is represented by I, while i indicates the current iteration number. The decreasing energy of the prey is denoted by the variable E_{p1}. During the iterative process, this value decreases linearly from 1.5 to 0, indicating the gradual depletion of the prey's energy.

$$E_{p0} = 2 * r - 1 \tag{22}$$

$$E_{p1} = c_1 * \left(1 - \frac{i}{I}\right) \tag{23}$$

Equations (19) and (20) are used to calculate the distance between the jackal and its prey as $X(i) - s1 * X_{prey}(i)$. The energy level of the prey controls the jackal's movement, which shifts its location either higher or lower depending on how far it is from the prey. The vector $s1$ employed in Equations (19) and (20) is a series of random numbers that complies with the Levy distribution, which is a specific type of probability distribution. This distribution is utilized to emulate the Levy movement, and it is multiplied by the Prey vector to determine the movement of the prey in a Levy fashion. The calculation of $s1$ shown in Equation (24).

$$s1 = 0.05 * LF(x) \tag{24}$$

The Levy Flight function (LF) is a mathematical function that simulates random movements in a search space. It is commonly used in optimization algorithms as it is used here. The process involves generating random numbers from the Levy distribution and using them to update the position of the search agent. The Levy distribution is a probability distribution with heavy tails, allowing for occasional large movements. This property is helpful in optimization because it enables search agents to explore distant areas of the search space that would be difficult to reach with small, incremental movements. LF can be calculated using Equation (25), where u, v u, v are a normal distribution function with a standard deviation of σ_u and σ_v such that $u = normal\left(0, \sigma_u^2\right)$ and $v = normal\left(0, \sigma_v^2\right)$. σ_u is calculated using the Equation (26).

$$LF(x) = \frac{0.05 * u}{v^{\frac{1}{\delta}}} \tag{25}$$

$$\sigma_u = \left[\frac{(1+\delta) * \sin\left(\frac{\pi\delta}{2}\right)}{\frac{1+\delta}{2} * \delta * 2^{\frac{\delta-1}{2}}} \right]^{\frac{1}{\delta}} \tag{26}$$

Equation (27) illustrates the position update of the male jackal and female jackal, which involves the averages Equations (19) and (20).

$$X(i+1) = \frac{X_a(i) + X_b(i)}{2} \tag{27}$$

4.3. Exploitation Phase

The simulation imitates the hunting behaviors of a dominant male golden jackal that takes the lead and guides the pack towards the food source to exploit the prey. The harassment of the prey by the jackals gradually reduces its ability to evade, enabling the male and female jackal pair to surround the prey discovered earlier. After being contained,

the jackals pounce on their target and devour it. In a mathematical model, the cooperative hunting behavior of jackals is represented in Equations (28) and (29), where i indicates the current iteration of the simulation. X_{prey} is the location vector of the prey, while $X_M(i)$ represents the location of the male jackal and $X_{FM}(i)$ represents the location of female jackal. $X_a(i)$ represents the revised location of the male jackal, and $X_b(i)$ represents the revised positions of the female jackal with respect to the prey. Equation (21) is employed to compute the evading energy of the prey, denoted as E_p. Equation (27) is then utilized to revise the positions of the jackals. In the exploitation phase, the function $s1$ is utilized in Equations (28) and (29) to promote exploration and prevent the algorithm from becoming trapped in local optima. Equation (24) is used to calculate $s1$, which helps to overcome sluggishness towards local optima, particularly in the final iterations. This element represents obstacles that hinder the jackals from moving towards the prey, such as those encountered in natural chasing paths. The function of $s1$ during the exploitation stage is to address these obstacles and facilitate the jackals' movement towards the prey.

$$X_a(i) = X_M(i) - E_p\left|s1 * X_M(i) - X_{prey}(i)\right| \tag{28}$$

$$X_b(i) = X_{FM}(i) - E_p\left|s1 * X_{FM}(i) - X_{prey}(i)\right| \tag{29}$$

4.4. Fitness and Transfer Function

Before computing fitness and updating it, the continuous values of the position matrix (X_{prey}) are converted into binary values using a transfer function. A sigmoid transfer function is used in this study, as shown in Equation (30). The reason for using this S-shaped transfer function is that it allows for a smooth and continuous transition from real-valued positions to binary values, which can help to avoid premature convergence and improve the search performance of the optimization algorithm.

$$TF = \frac{1}{1 + e^{-X}} \tag{30}$$

In this equation, X represents the position value in the position matrix (X_{prey}) before being converted to binary. The sigmoid function maps the continuous value of X to a value having 0 and 1, which can then be used to determine the corresponding binary value. The purpose of this conversion is to ensure that the position values are binary and can be used to calculate the fitness value of the prey.

The fitness in this context refers to the prediction error of a machine learning (ML) classifier. It is determined by comparing the actual output of the classifier with its estimated output. To train the classifier, a 0.2 data split size is used, meaning that 20% of the data is held out for testing while the remaining 80% is used for training. The fitness is calculated using the Equation (31), where k is a value that ranges from 1 to m (the number of testing observations) and $Err(k)$ is the prediction error for the kth observation. The summation is divided by m to obtain an average prediction error.

$$fitness = \sum_{i=1}^{m} \frac{Err(k)}{m} \tag{31}$$

The algorithm maintains two variables for the updating of fitness; the variables $MaleJackalscore$ and $FemaleJackalscore$ represent the fitness scores of the best male and female jackals found so far during the optimization process. $fitness$ can be assumed as old fitness; $MaleJackalscore$ and $FemaleJackalscore$ can be assumed as new fitness. If the $fitness$ of a jackal is lower than the current $MaleJackalscore$, it means that the jackal has a better $fitness$ than the current male jackal, and thus, its position and score will replace the current male jackal's position and score. On the other hand, if the $fitness$ of a jackal is higher than the $MaleJackalscore$ but lower than the $FemaleJackalscore$, it means that the jackal has a better fitness than the current female jackal, but not better than the male jackal.

In this case, its position and score will replace the current female jackal's position and score. After the fitness calculation, the fitness stored as shown in Equation (32), where the fitness array is denoted as f_i, which consists of p elements $f_1, f_2, f_3, \ldots, f_p$.

$$f_i = (f_1, f_2, f_3, \ldots, f_p) \tag{32}$$

In each iteration of the algorithm, a random value between -1 and 1 is assigned to the initial energy E_{p0}. The value of E_{p0} is an indicator of the prey's physical strength, where a decrease from 0 to -1 indicates a decline in the prey's strength. An elevation from 0 to 1 denotes a boost in the prey's strength, whereas a decrease in E_p is observed during the iterative process, as shown in Figure 1. If the magnitude of E_p is greater than 1, it means that the jackal pairs are searching for prey in different areas, which suggests that the algorithm is in an exploration phase. On the other hand, if the magnitude of E_p is less than 1, the algorithm switches to an exploitation phase and starts attacking the prey (Algorithms 1).

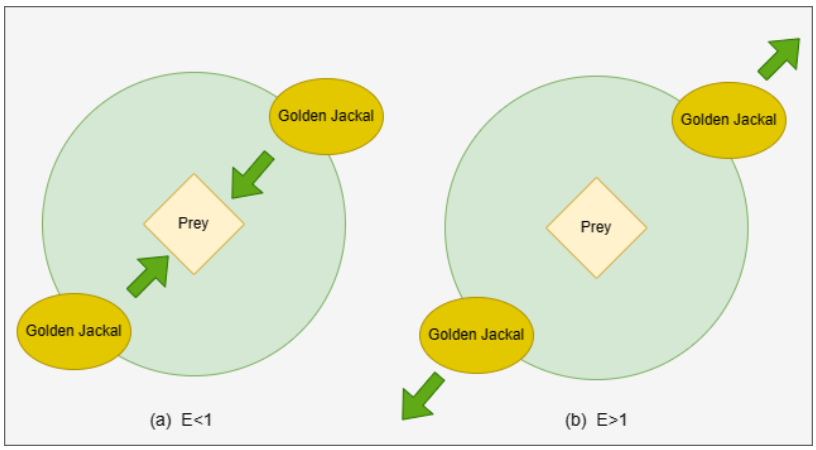

Figure 1. Searching and Attacking.

Algorithms 1 FSGJO

Initialize prey population randomly, $X_i = (i = 1, 2, \ldots, P)$
while $(i < I)$
  Let, Male Jackal Position be X_a
  Let, Female Jackal Position be X_b
  Determine the preys' fitness value
  if $(fitness < MaleJackalscore)$
    $MaleJackalscore = fitness$
  if $(fitness > MaleJackalscore$ and $fitness < FemaleJackalscore)$
    $FemaleJackalscore = fitness$
 for (each prey)
    Using Equations (21)–(23) update the evading energy E_p)
    Using Equations (24) and (25) Update $s1$
    if $(E \geq 1)$ (Exploration phase)
      Using Equations (19), (20) and (27) Update the prey position
    if $(E < 1)$ (Exploration phase)
      Using Equations (27)–(29) Update the prey position
  Update Jackal Position, $X(i) = \frac{X_a + X_b}{2}$
  Using transfer function to convert continuous values of X_i i.e., position, in binary values using Equation (30)
  end for
  i++
end while
Return Male Jackal Position X_a

The FSGJO algorithm is an optimization algorithm in metaheuristic form that works on the hunting pattern of golden jackals. After randomly initializing a population of prey,

the algorithm proceeds to search for the ideal solution through a series of iterations. The algorithm employs the idea of jackals, wherein the male jackal denotes the best solution found thus far, while the female jackal represents the second-best solution. The algorithm updates the position and evading energy of each prey based on certain equations and then performs an exploration or exploitation phase depending on the value of the evading energy. The algorithm updates the jackal position by taking the average of the male and female positions. Then it converts the continuous values of the prey positions into binary values using a transfer function. The algorithm continues for a specified repetitions and returns the male jackal position, representing the best solution found by the algorithm. The detailed explanation of FSGJO algorithm is presented in Algorithms 1. A flowchart depicting it is shown in Figure 2.

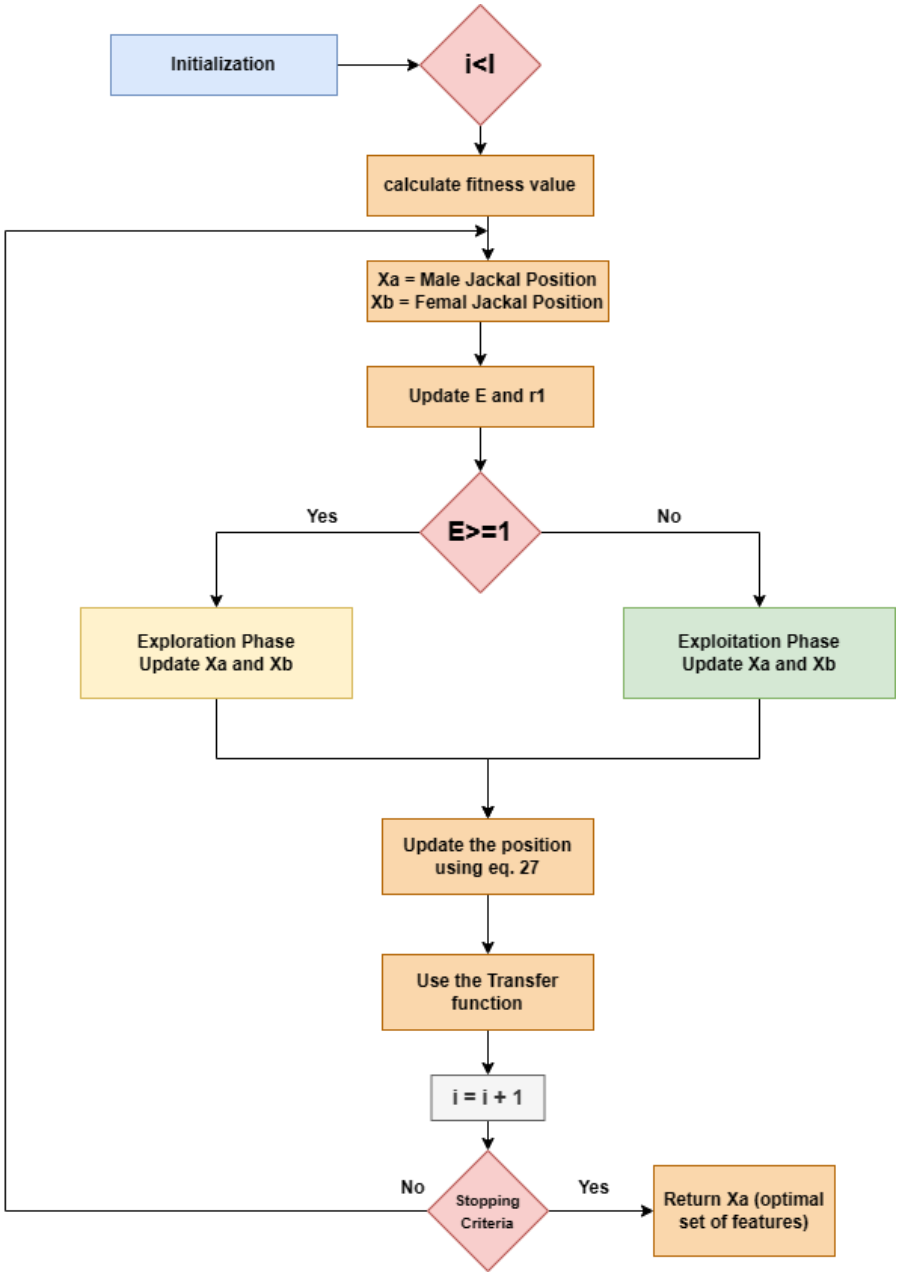

Figure 2. Flowchart for FSGJO.

5. Results

This section provides information on the datasets utilized in the experiment, the experimental setup, and the analysis of the obtained results.

5.1. Datasets

Most of these datasets are publicly available and have been used in various software engineering and machine learning research studies. Some of these are commonly used benchmark datasets, and their sources can be found in multiple academic publications or online repositories. The PROMISE repository provides a collection of datasets for various software engineering tasks, including software fault prediction. These datasets are primarily meant for research purposes and are frequently employed to assess the efficacy of different software fault prediction models. The datasets in the PROMISE repository are sourced from various software projects and programming languages. Each dataset usually contains additional metrics and features that characterize the analyzed software, along with details on the occurrence or non-occurrence of faults in the software. Some examples of the metrics and features that are in these datasets are:

- LOC (Lines of Code): This metric measures the number of lines of code in the software being analyzed.
- Cyclomatic Complexity: This metric measures the complexity of the software's control flow and can help identify potential trouble spots.
- Code Churn: This metric measures the software's change over time and can help identify modules or components that may be more prone to faults.
- Code Coverage: This metric measures the extent to which the software's code has been tested and can help identify code areas that may be more likely to contain faults.
- Halstead's Complexity Measures: These metrics measure various aspects of the complexity of the software's code, such as the number of distinct operators and operands, and can help identify potential trouble spots.

Each dataset in the PROMISE repository typically includes a description of the software project, as well as information on the available metrics and features. The datasets may also include information about the presence or absence of faults in the software, such as the number of bugs that were discovered during testing or the number of incidents that were reported by users. Researchers can use these datasets to train and test different software fault prediction models. To assess the efficacy of diverse software fault prediction approaches and discover scopes for further enhancement, researchers can analyze the performance of various models on a common dataset. Here, 12 datasets from the PROMISE repository by NASA have been used. The datasets are KC1, PC5, MC1, JM1, PC1, MW1, PC2, KC3, PC4, CM1, and MC2. Table 1 provides the specifics of the datasets.

Table 1. Detail of the datasets.

S. No.	Datasets	Number of Instances	Number of Features
1	MC1	1988	39
2	MC2	125	40
3	MW1	253	38
4	PC1	705	38
5	PC2	745	37
6	PC3	1077	38
7	PC4	1287	38
8	PC5	1711	39
9	CM1	327	38
10	KC1	1183	22
11	KC3	194	40
12	CM1	327	38

In dimensionality reduction, there are some issues with the dataset such as correlation and collinearity. Correlation is a statistical measure that describes the strength and direction of a relationship between two variables. Correlation can be used to explore the relationship between any two quantitative variables. Feature selection can be used to deal with the correlation problem in data analysis and modeling. Feature selection is a technique that aims to select a subset of the most important features from a set of features in a dataset. By selecting only the most important features, we can reduce the impact of correlated variables and improve the performance of our models. The algorithms such as FSGE, FSPSO, FSDE, and FSACO and the proposed FSGJO algorithm used in the manuscript can effectively deal with correlation and other complex dependencies between features. Table 2 shows the correlation data for different datasets. It ranges from −1 to +1, where −1 indicates a perfect negative correlation (as one variable increases, the other decreases), +1 indicates a perfect positive correlation (as one variable increases, the other also increases), and 0 indicates no correlation between the variables.

Table 2. Correlation Data for different Datasets.

Sr. No.	Datasets	Avg Mean	Avg Min	Avg Std	Avg (25%)	Avg (50%)	Avg (75%)	Max
1	PC1	0.378607	−0.33978	0.333734	0.13995	0.472284	0.584182	1
2	PC2	0.406	−0.334	0.361	0.147	0.539	0.627	1
3	PC3	0.28585	−0.32153	0.299	0.122	0.287	0.449	1
4	PC4	0.28585	−0.3215	0.299	0.122	0.287	0.449	1
5	PC5	0.309287	−0.25837	0.318001	0.107005	0.337127	0.505981	1
6	JM1	0.523668	−0.26472	0.297546	0.434064	0.58751	0.695541	1
7	KC1	0.602172	−0.31984	0.311581	0.583292	0.694918	0.7534	1
8	KC3	0.401659	−0.49557	0.381206	0.125805	0.540514	0.642491	1
9	MW1	0.3446	−0.438	0.3462	0.0564	0.4107	0.5741	1
10	MC1	0.329962	−0.2981	0.34041	0.08559	0.42838	0.55244	1
11	MC2	0.422839	−0.36885	0.364055	0.262204	0.565452	0.643115	1
12	CM1	0.447366	−0.39526	0.34699	0.22507	0.57133	0.64822	1

For example, the first row of Table 2 shows the correlation data for the dataset named "PC1". The average correlation value for this dataset is 0.378607, indicating a moderate positive correlation between the variables. The minimum correlation value is −0.33978, indicating some negative correlation between the variables, and the maximum correlation value is 1, indicating a perfect positive correlation between the variables. The standard deviation of correlation values for this dataset is 0.333734, indicating that the correlation values vary widely in the dataset. The value in the max column is 1, which suggests that at least one pair of variables has a perfect correlation. Similarly, each row in the table shows the correlation data for a different dataset. The correlation values can help identify patterns in the data, such as strong positive or negative correlations, weak correlations, or no correlations. This information can be useful for statistical analysis, such as identifying which variables are most strongly related to each other, or for modeling, such as using the correlation data to make predictions or develop models. Overall, the correlation data in this table provides valuable information about the relationships between variables in different datasets, which can help researchers and analysts gain insights into the data and make informed decisions based on the results.

Collinearity is a problem that occurs when two or more features in a dataset are highly correlated. In the context of feature selection, collinearity can make it difficult to identify the most important features. This is because the coefficient estimates for all of the correlated features may be large, even if only one of the features is truly independent. There are

a number of ways to deal with collinearity. One way is to use a correlation matrix to identify correlated features. The correlation matrix for dataset MC1 and JM1 is shown in Figures 3 and 4, respectively. Once you have identified correlated features, you can use feature selection to remove them from the dataset. There are a number of different feature selection methods available. Feature selection can be a useful tool for dealing with collinearity. By removing correlated features from the dataset, you can improve the stability of the coefficient estimates and the accuracy of the model, as we have used the different feature selection methods such as FSGA, FSPSO, FSDE, and FSACO and the newly proposed FSGJO which does the work efficiently.

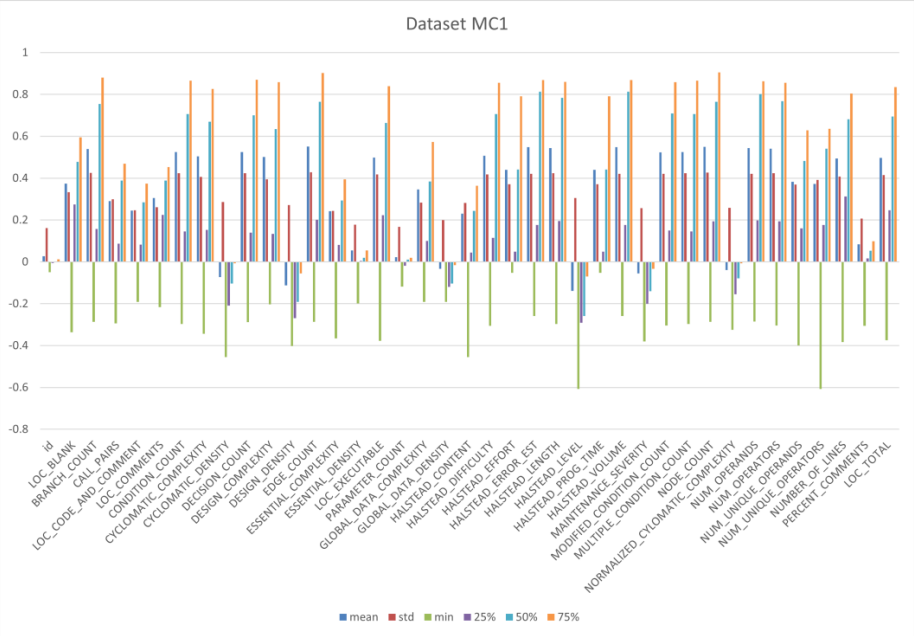

Figure 3. Correlation Matrix graph for MC1 Dataset.

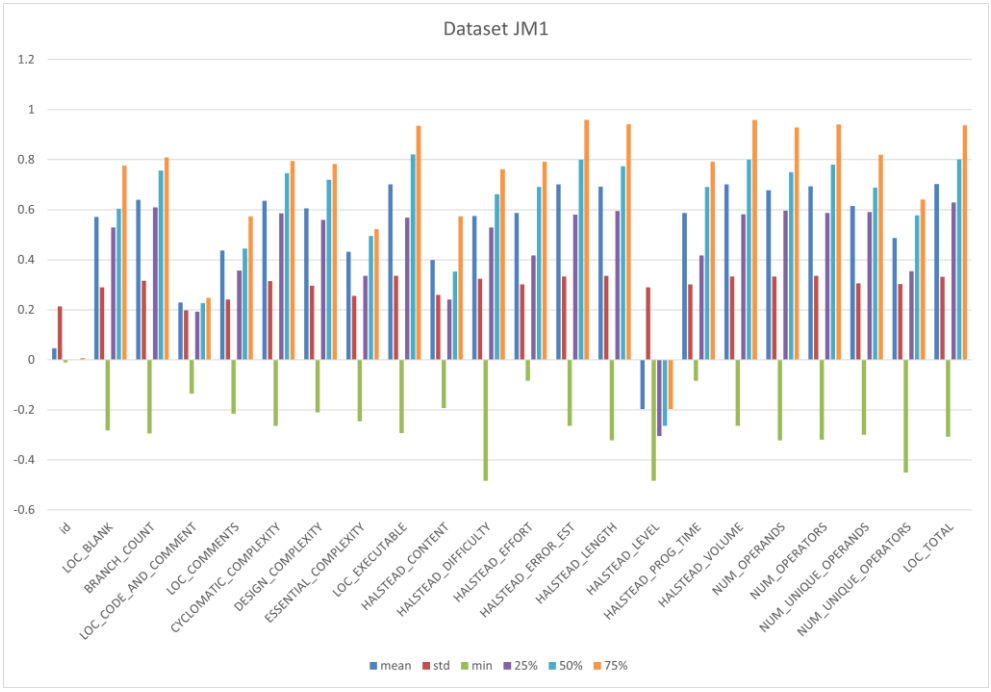

Figure 4. Correlation Matrix graph for JM1 Dataset.

5.2. Experimental Condition

In the context of the study, the trial was conducted using VS Code and the 3.9.12 version of Python. The laptop utilized for the experiment was equipped with an AMD Ryzen 75,000 series processor, a clock speed of 1.80 GHz, 16 GB of RAM, and AMD Radeon graphics. The parameters of the experiment such as β were set to 1.5; population size was set to 30, and the maximum iteration was set to 200.

5.3. Experimental Analysis

To predict software faults, 12 different datasets that were used are listed in Table 2. The classifiers employed for the experiments are DT, KNN, NB, and QDA. Then, randomly split the datasets into training and testing sets, maintaining a ratio of 80:20, respectively. To maintain the consistency of each algorithm's performance, we conducted 10 runs of the experiments. The average accuracy results obtained from FSGJO and the other FS models are presented in Table 3.

Table 3. Comparison between different FS Algorithms.

S. No.	Datasets	Classifier	Without FS (%)	FSGA (%)	FSPSO (%)	FSDE (%)	FSACO (%)	FSGJO (%)
1	PC1	KNN	89.36	93.67	90.08	93.4	93.71	94.42
		DT	88.66	95.01	92.05	95.02	93.79	95.03
		NB	87.32	91.12	89.56	90.46	92.26	92.67
		QDA	86.25	93.62	89.27	93.84	92.19	94.32
No. of features selected			38	19.1	16.2	18.9	10.8	13
2	PC2	KNN	96.46	97.66	97.54	97.34	97.09	98.65
		DT	95.23	98.41	96.15	98.11	97.52	98.65
		NB	93.26	96.89	95.48	96.13	97.23	96.98
		QDA	97.23	98.85	97.83	97.83	98.21	97.98
No. of features selected			37	14.7	15.2	17.1	10.7	11
3	PC3	KNN	82.14	86.43	84.67	85.39	86.17	87.03
		DT	78.07	86.93	82.19	86.73	84.34	87.03
		NB	68.89	86.58	80.38	86.18	87.8	87.3
		QDA	62.4	86.54	83.83	86.67	86.09	87.5
No. of features selected			38	16.6	13.5	17.2	11.6	11.4
4	PC4	KNN	84.05	90.18	86.36	87.06	91.61	90.69
		DT	91.9	93.52	92.64	93.52	92.4	93.02
		NB	86.28	91.95	89.1	91.47	91.04	91.86
		QDA	47.76	91.28	86.89	92.84	91.28	93.41
No. of features selected			38	17.6	14.4	18.6	14.2	14.6
5	PC5	KNN	67.6	75.36	71.8	75.36	76.58	78.42
		DT	72.95	77.37	73.35	77.18	75.61	77.84
		NB	70.45	71.75	70.28	71.64	72.91	72.99
		QDA	69.93	72.75	70.85	72.29	71.1	73.46
No. of features selected			39	18.4	15.7	19.3	14.8	17
6	JM1	DT	73.53	77.81	75.6	76.63	79.62	78.16
		KNN	69.49	78.63	72.49	73.39	79.59	79.6
		NB	78.01	79.84	79.15	79.62	79.89	79.89
		QDA	75.85	79.71	79.04	79.82	79.78	79.82
No. of features selected			22	4.9	8.9	10.3	3	6
7	KC1	KNN	69.26	76.46	76.46	76.33	77.59	78.05
		DT	72.51	77.6	73.21	76.3	76.48	77.79
		NB	74.62	77.32	76.21	77.32	77.47	77.63
		QDA	74.62	78.01	76.92	77.39	77.58	78.48

Table 3. *Cont.*

S. No.	Datasets	Classifier	Without FS (%)	FSGA (%)	FSPSO (%)	FSDE (%)	FSACO (%)	FSGJO (%)
No. of features selected			22	8.2	8.3	9.4	4.5	8
8	KC3	KNN	74.63	79.89	76.51	79.32	86.29	82.05
		DT	76.29	89.54	81.3	87.96	85.31	89.74
		NB	66.76	76.29	71.45	76.51	79.39	76.92
		QDA	76.29	86.29	79.47	87.96	86.15	89.74
No. of features selected			40	17.7	16.9	18.8	8.9	16.6
9	CM1	KNN	75.67	86.28	83.43	85.03	87.24	89.39
		DT	80.03	89.07	83.28	88.49	87.37	89.39
		NB	77.37	83.34	81.97	83.28	84.45	84.46
		QDA	83.21	88.84	83.79	88.84	88.81	90.9
No. of features selected			38	18.2	14.5	17.8	12.1	15.8
10	MC1	KNN	96.37	97.63	97.46	97.48	98.10	97.73
		DT	97.64	98.47	98.39	98.57	98.24	98.74
		NB	95.64	97.61	96.21	97.62	97.64	97.73
		QDA	97.39	97.64	97.39	97.64	97.64	97.73
No. of features selected			39	19.2	12.4	19.2	13.4	13.2
11	MC2	KNN	75	87.46	79	85.12	89.12	92
		DT	68	90.78	75.21	89.26	85	89.26
		NB	93	95.56	92.71	93.12	95	96
		QDA	83	95.12	88.34	95.12	95.12	96
No. of features selected			40	18.4	17.2	18.4	7.2	8
12	MW1	KNN	78.34	87.35	84.61	85.56	86.57	88.27
		DT	74.41	87.74	82.64	87.15	85.19	85.29
		NB	76.37	83.42	78.78	82.25	87.16	88.27
		QDA	80.49	88.52	84.21	86.56	90.29	92.14
No. of features selected			38	13.5	12.9	17.2	8.7	7.8

The performance of the novel FSGJO algorithm is evaluated against various FS techniques, including FSPSO, FSGA, FSACO, and FSDE, using a set of 12 datasets obtained from NASA's open repository. Table 3 provides the performance comparison of various feature selection techniques on different datasets using different classifiers. The table represents the average classification accuracy of various classifiers applied to different datasets with and without feature selection. In addition, the table also presents the mean number of features chosen by each FS technique. The classifiers were tested on various datasets using diverse feature selection methods mentioned previously. From the results, FSGJO has performed well in most of the datasets except for a few of the datasets. For the PC4 dataset, the highest accuracy has been achieved by the FSGA model, but the difference between the accuracy of both the FSGJO and FSGA models is very minor. For the MC1 and MC2 datasets, the average accuracy of FSGA, FSDE, and FSPSO for the QDA classifier is the same. Similarly, for other datasets the accuracy of models is somewhat the same and somewhat different, less or more with each other. However, for the majority of the cases FSGJO has greater average accuracy.

The fitness error plot of the four classifiers—DT, KNN, NB, and QDA—is shown in Figures 5–8, respectively. It includes the error plots of each FS model—FSGA, FSPSO, FSDE, FSACO, and FSGJO. Each figure contains the plots for all the 12 datasets. From the error plot, it can be seen that for many times the plot for FSGJO is lower, but for some it coincides with other FS models, and for some it is above the other. In Figure 5 (DT classifier), the fitness plots of FSGA and FSGJO coincide with each other at 165 iterations in the PC3 dataset. It is similar for FSACO and FSGJO in the MW1 dataset after 145 iterations. In the KNN classifier, the coincidences of the error plot occur in CM1 and PC2 between

FSACO, FSDE, and FSGJO after 50 and 150 iterations, respectively. In the NB classifier, the coincidences of the error plot occur in the KC3 dataset between FSACO and FSDE, in the MC2 and MC1 dataset between FSACO and FSGJO, and in the PC3 dataset between FSACO and FSGJO for some number of iterations. Lastly, in the QDA classifier, the coincidences of the error plot occurred in MC1 datasets for all classifiers in the MC2 dataset between FSACO, FSDE, and FSGJO. For the rest of the other datasets and classifiers, the error plot of FSGJO is less than the others which shows that FSGJO performs better than the other models.

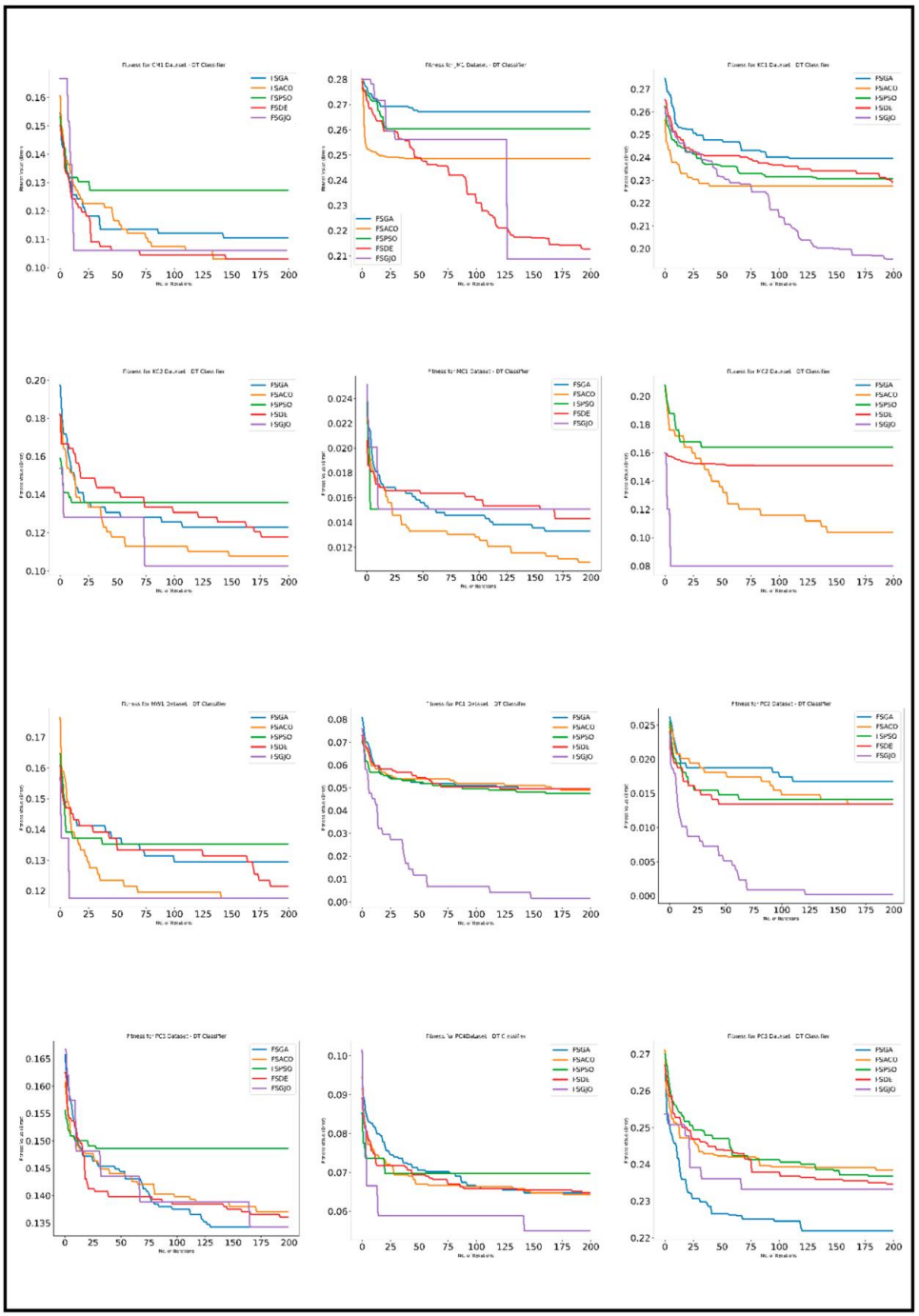

Figure 5. Fitness Error Plot for DT.

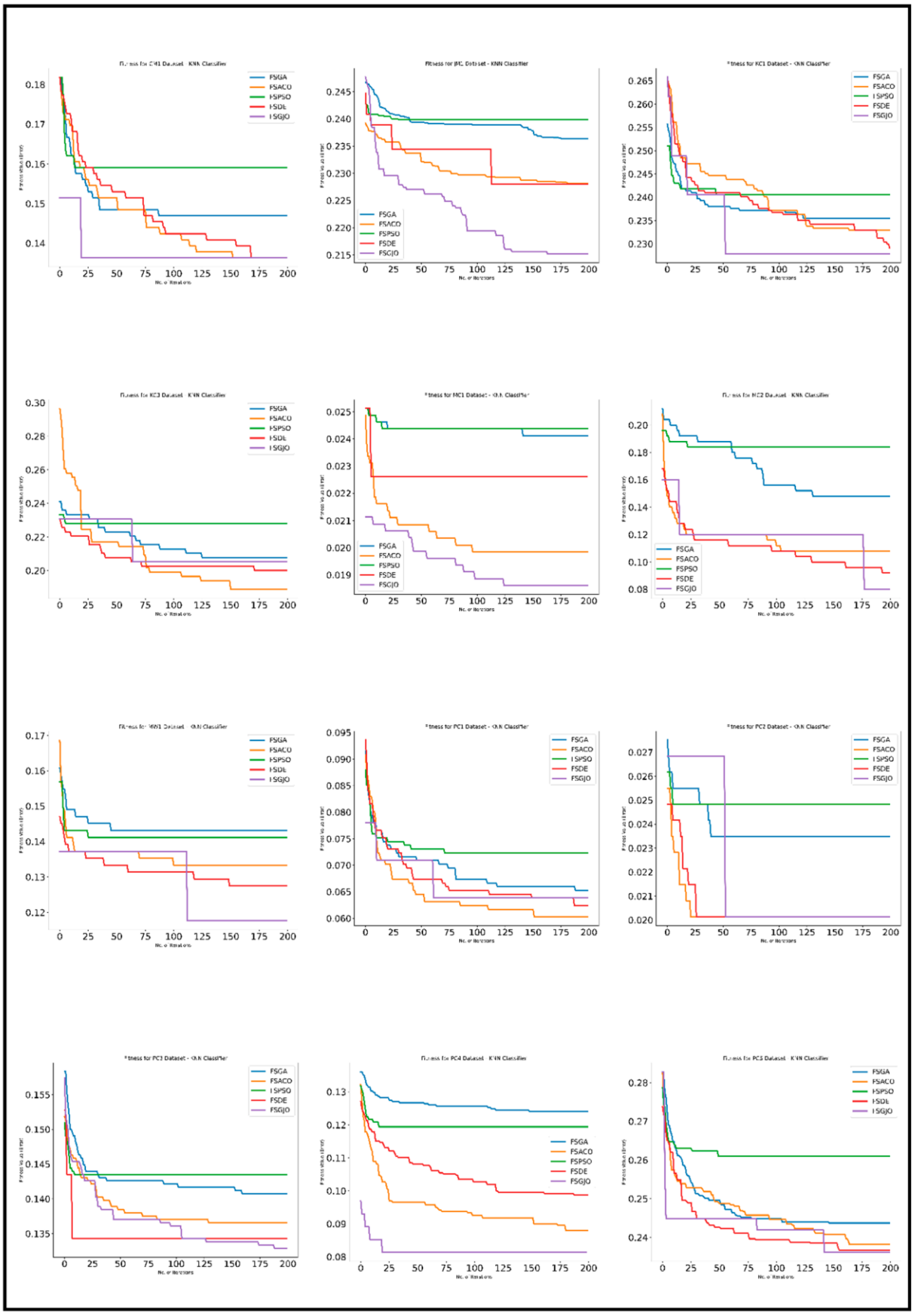

Figure 6. Fitness Error Plot for KNN.

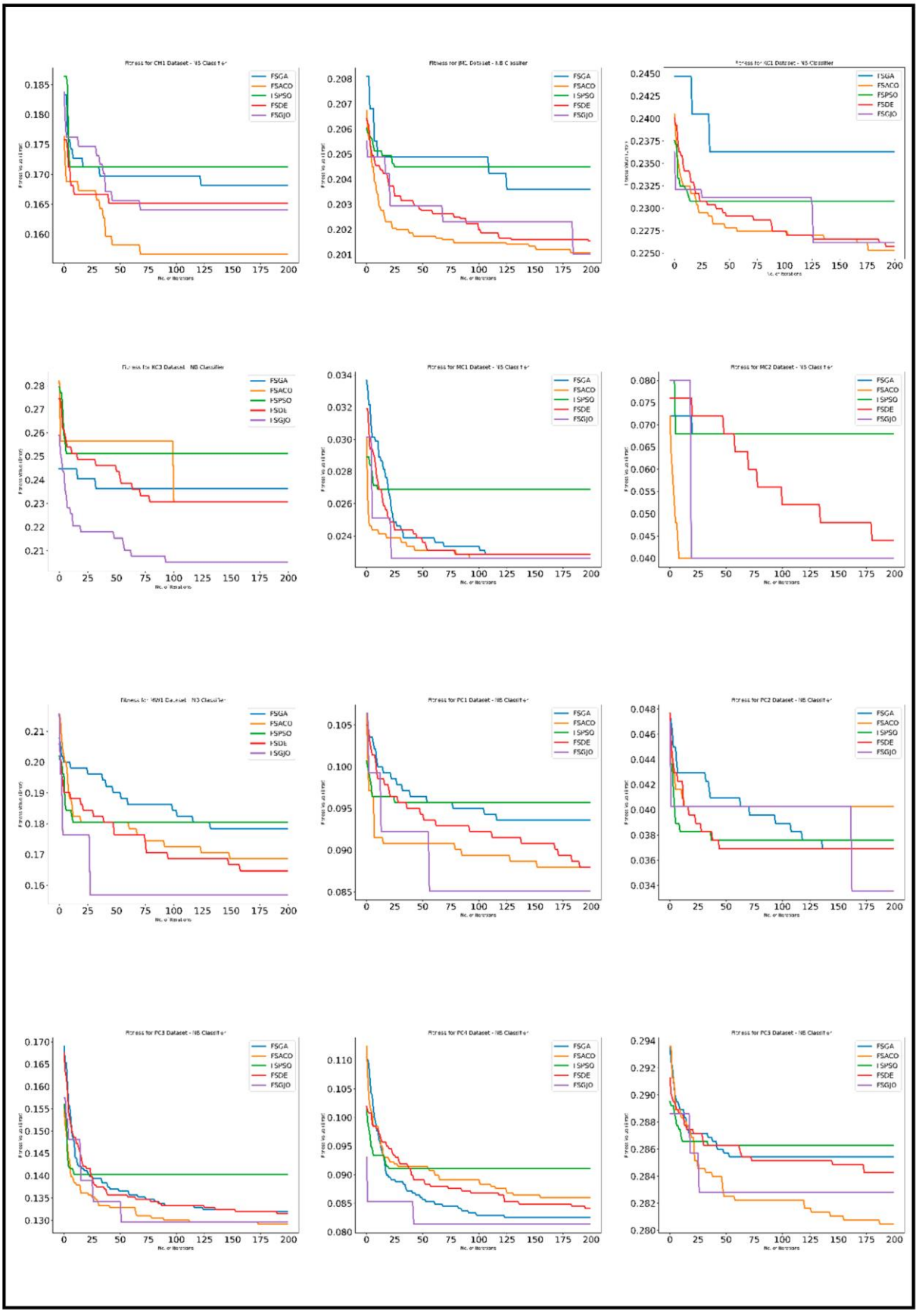

Figure 7. Fitness Error Plot for NB.

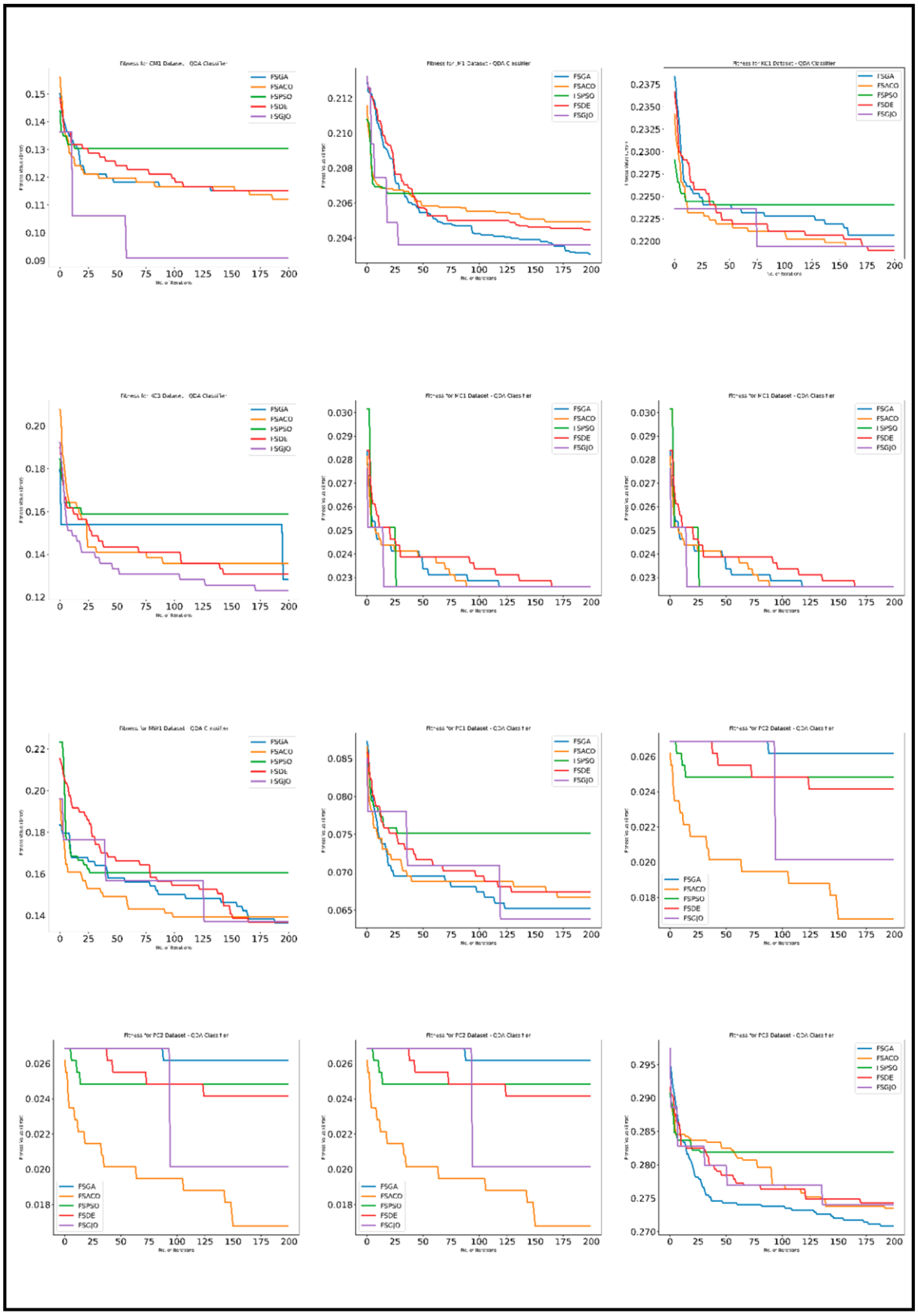

Figure 8. Fitness Error Plot for QDA.

The parameters utilized in various FS models are listed in Table 4. The algorithms utilized in the study employ different parameters. The number of populations is set to 30 for all models, and the max number of iterations is 200. The value of δ is 1.5 in GJO. To specify the parameters for ACO, the values of alpha, beta, and rho are set to 1, 0.1, and 0.2, respectively. In GA, the mutation rate (MR) utilized is 0.01, and the crossover rate (CR) is 0.8. The utilization of 0.9 as the crossover rate (CR) and 0.8 as the scaling factor (SF) is common in Differential Evolution (DE). The initial weights W_{min} and W_{max} in PSO are set to 0.4 and 0.9, respectively.

Table 4. Parameters used in different FS models.

Parameters	GA	PSO	DE	ACO	GJO
No. of iterations	200	200	200	200	200
Population Size	30	30	30	30	30
W_{max}	-	0.9	-	-	-
W_{min}	-	0.4	-	-	-
SF	-	-	0.8	-	-
c1	-	2	-	-	-
CR	0.8	-	0.9	-	-
MR	0.01	-	-	-	-
c2	-	2	-	-	-
α (alpha)	-	-	-	1	-
β (beta)	-	-	-	0.1	-
ρ (rho)	-	-	-	0.2	-
δ	-	-	-	-	1.5

6. Statistical Analysis

Statistical analysis [51] is an important component of machine learning (ML), as it helps to make sense of data by identifying patterns and relationships. By using the same parameters, the proposed model can be compared with other models in terms of their performance. Some common statistical techniques used in ML include regression analysis, cluster analysis, principal component analysis (PCA), hypothesis testing, and Bayesian analysis. Overall, statistical analysis is an essential tool for understanding and making sense of data in machine learning. In statistical hypothesis testing, there are two main types of tests: parametric (using parameters) and nonparametric (using a hypothesis). Parametric statistical testing is a type of statistical analysis that assumes that the data being analyzed follows a particular probability distribution, most commonly the normal distribution. This assumption allows for the use of a range of statistical tests that are more powerful than their non-parametric counterparts. The use of parametric tests requires that several assumptions are met, including the assumptions of normality, homogeneity of variance, and independence of observations. When conducting a parametric test, it is crucial to ensure that the assumptions are met, as any violations of these assumptions can lead to inaccurate outcomes and conclusions that are not valid. Therefore, it is imperative to verify these assumptions beforehand and to consider utilizing non-parametric tests if they are not met. Non-parametric statistical testing is a form of statistical analysis that does not rely on a particular probability distribution assumption for the data being analyzed. Instead, non-parametric tests are based on the ranks or orderings of the data, making them more robust to violations of assumptions and more widely applicable than parametric tests. Non-parametric tests do not require assumptions of normality, homogeneity of variance, or independence of observations, making them more versatile than parametric tests. However, they may be less powerful than parametric tests when the assumptions of the parametric tests are met. Selecting an appropriate statistical test that is tailored to the research question and the characteristics of the data being analyzed is crucial. Here, the Friedman test has

been used, which is a non-parametric statistical test that is used to compare three or more related groups.

The Friedman test is utilized to determine if there are any noteworthy disparities between groups by comparing their average ranks. The test's null hypothesis suggests that there are no differences between the groups, while the alternative hypothesis proposes that at least one group displays a significant difference from the others. The test statistic used in the Friedman test is based on the chi-squared distribution, and the significance level is determined using the appropriate critical values from the chi-squared distribution table. In cases where the computed test statistic is higher than the critical value, the null hypothesis is refuted, signifying a significant distinction between the groups. One important consideration when using the Friedman test is that it is an omnibus test, meaning that it only determines whether there is a significant difference between the groups as a whole. Additional tests, such as post hoc analyses, may be necessary to identify the particular groups that exhibit significant differences from one another.

In hypothesis testing, the null hypothesis (H_0) postulates that there is no significant disparity between the models, whereas the alternative hypothesis (H_1) proposes the contrary. When the p-value is less than the significance level, it means there is a difference between two or more models, and we can reject the null hypothesis. The Friedman test assigns a rank to each model based on its classification performance in the experiment. The models are ranked from the one with the lowest number to the one with the highest number, with the highest rank assigned to the one with the lowest number and the lowest rank assigned to the one with the highest number. Table 5 presents the results of evaluating various models (FSACO, FSDE, FSGA, FSGJO, FSPSO, and Without FS) with different classifiers (KNN, DT, NB, and QDA), showing the $AvgRankModels$ calculated with Equation (33). Table 5 presents the $AvgRankDatasets$ obtained by evaluating the mean of all ranks for each associated model (Without FS, FSGA, FSPSO, FSDE, FSACO, and FSGJO) across all datasets using Equation (34). The computation of $AvgRankDatasets$ involves adding up the ranks of all classification models used and dividing the sum by the total number of models. In Table 6, we report the average ranks of the feature selection (FS) models employed in our experiment. To calculate the average rank performance of a group of models and datasets, add up the mean rank of each one and divide by the total number of models.

$$AvgRankModels = \frac{total\ sum\ of\ rank\ of\ different\ classifiers}{total\ no.\ of\ classifiers} \tag{33}$$

$$AvgRankDatasets = \frac{AvgRankModels}{total\ no.\ of\ datasets} \tag{34}$$

Table 5. FS Algorithm ranks for 12 datasets using the Friedman Test.

S. No.	Datasets	Classifier	Without FS (%)	FSGA (%)	FSPSO (%)	FSDE (%)	FSACO (%)	FSGJO (%)
1	PC1	KNN	89.36 (6)	93.67 (3)	90.08 (5)	93.4 (4)	93.71 (2)	94.42 (1)
		DT	88.66 (6)	95.01 (3)	92.05 (5)	95.02 (2)	93.79 (4)	95.03 (1)
		NB	87.32 (6)	91.12 (3)	89.56 (5)	90.46 (4)	92.26 (2)	92.67 (1)
		QDA	86.25 (6)	93.62 (3)	89.27 (5)	93.84 (2)	92.19 (4)	94.32 (1)
Avg. Rank of Models			6	3	5	3	3	1
2	PC2	KNN	96.46 (6)	97.66 (2)	97.54 (3)	97.34 (4)	97.09 (5)	98.65 (1)
		DT	95.23 (6)	98.41 (2)	96.15 (5)	98.11 (3)	97.52 (4)	98.65 (1)
		NB	93.26 (6)	96.89 (3)	95.48 (5)	96.13 (4)	97.23 (1)	96.98 (2)
		QDA	97.23 (5)	98.85 (2)	97.83 (4)	97.83 (4)	98.21 (3)	97.98 (1)
Avg. Rank of Models			5.75	2.25	4.25	3.75	3.25	1.25

Table 5. *Cont.*

S. No.	Datasets	Classifier	Without FS (%)	FSGA (%)	FSPSO (%)	FSDE (%)	FSACO (%)	FSGJO (%)
3	PC3	KNN	82.14 (6)	86.43 (2)	84.67 (5)	85.39 (4)	86.17 (3)	87.03 (1)
		DT	78.07 (6)	86.93 (2)	82.19 (5)	86.73 (3)	84.34 (4)	87.03 (1)
		NB	68.89 (6)	86.58 (3)	80.38 (5)	86.18 (4)	87.8 (2)	87.3 (1)
		QDA	62.4 (6)	86.54 (3)	83.83 (5)	86.67 (2)	86.09 (4)	87.5 (1)
Avg. Rank of Models			6	2.50	5	3.25	3.25	1
4	PC4	KNN	84.05 (6)	90.18 (3)	86.36 (5)	87.06 (4)	91.61 (1)	90.69 (2)
		DT	91.90 (5)	93.52 (1)	92.63 (3)	93.52 (1)	92.40 (4)	93.02 (2)
		NB	86.28 (6)	91.95 (1)	89.10 (5)	91.47 (3)	91.04 (4)	91.86 (2)
		QDA	47.76 (5)	91.28 (3)	86.89 (4)	92.84 (2)	91.28 (3)	93.41 (1)
Avg. Rank of Models			5.50	2	4.25	2.50	3	1.75
5	PC5	KNN	67.6 (5)	75.36 (3)	71.80 (4)	75.36 (3)	76.58 (2)	78.42 (1)
		DT	72.95 (6)	77.37 (2)	73.35 (5)	77.18 (3)	75.61 (4)	77.84 (1)
		NB	70.45 (6)	71.75 (3)	70.28 (5)	71.64 (4)	72.91 (2)	72.99 (1)
		QDA	69.93 (6)	72.75 (2)	70.85 (5)	72.29 (3)	71.10 (4)	73.46 (1)
Avg. Rank of Models			5.75	2.50	4.75	3.25	3	1
6	JM1	DT	73.53 (6)	77.81 (3)	75.60 (5)	76.63 (4)	79.62 (1)	78.16 (2)
		KNN	69.49 (6)	78.63 (3)	72.49 (5)	73.39 (4)	79.59 (2)	79.60 (1)
		NB	78.01 (5)	79.84 (2)	79.15 (4)	79.62 (3)	79.89 (1)	79.89 (1)
		QDA	75.85 (5)	79.71 (3)	79.04 (4)	79.82 (1)	79.78 (2)	79.82 (1)
Avg. Rank of Models			5.50	2.75	4.50	3	1.50	1.25
7	KC1	KNN	69.26 (5)	76.46 (3)	76.46 (3)	76.33 (4)	77.59 (2)	78.05 (1)
		DT	72.51 (6)	77.60 (2)	73.21 (5)	76.30 (4)	76.48 (3)	77.79 (1)
		NB	74.62 (5)	77.32 (3)	76.21 (4)	77.32 (3)	77.47 (2)	77.63 (1)
		QDA	74.62 (6)	78.01 (2)	76.92 (5)	77.39 (4)	77.58 (3)	78.48 (1)
Avg. Rank of Models			5.50	2.50	4.25	3.75	2.50	1
8	KC3	KNN	74.63 (6)	79.89 (3)	76.51 (5)	79.32 (4)	86.29 (1)	82.05 (2)
		DT	76.29 (6)	89.54 (2)	81.30 (5)	87.96 (3)	85.31 (4)	89.74 (1)
		NB	66.76 (6)	76.29 (5)	71.45 (4)	76.51 (2)	79.39 (3)	76.92 (1)
		QDA	76.29 (6)	86.29 (3)	79.47 (5)	87.96 (2)	86.15 (4)	89.74 (1)
Avg. Rank of Models			6	3.25	4.75	2.75	3	1.25
9	CM1	KNN	75.67 (6)	86.28 (3)	83.43 (5)	85.03 (4)	87.24 (2)	89.39 (1)
		DT	80.03 (6)	89.07 (2)	83.28 (5)	88.49 (3)	87.37 (4)	89.39 (1)
		NB	77.37 (6)	83.34 (3)	81.97 (5)	83.28 (4)	84.45 (2)	84.46 (1)
		QDA	83.21 (5)	88.84 (2)	83.79 (4)	88.84 (2)	88.81 (3)	90.90 (1)
Avg. Rank of Models			5.75	2.50	4.75	3.25	2.75	1
10	MC1	KNN	96.37 (6)	97.63 (3)	97.46 (5)	97.48 (4)	98.10 (1)	97.73 (2)
		DT	97.64 (2)	98.47 (4)	98.39 (5)	98.57 (3)	98.24 (6)	98.74 (1)
		NB	95.64 (6)	97.61 (4)	96.21 (5)	97.62 (3)	97.64 (2)	97.73 (1)
		QDA	97.39 (3)	97.64 (2)	97.39 (3)	97.64 (2)	97.64 (2)	97.73 (1)
Avg. Rank of Models			4.25	3.25	4.50	3.00	2.75	1.25
11	MC2	KNN	75 (6)	87.46 (3)	79 (5)	85.12 (4)	89.12 (2)	92 (1)
		DT	68 (5)	90.78 (1)	75.21 (4)	89.26 (2)	85 (3)	89.26 (2)
		NB	93 (6)	95.56 (2)	92.71 (5)	93.12 (4)	95 (3)	96 (1)
		QDA	83 (4)	95.12 (2)	88.34 (3)	95.12 (2)	95.12 (2)	96 (1)
Avg. Rank of Models			5.25	2.00	4.25	3.00	2.50	1.25
12	MW1	KNN	78.34 (6)	87.35 (2)	84.61 (5)	85.56 (4)	86.57 (3)	88.27 (1)
		DT	74.41 (6)	87.74 (1)	82.64 (5)	87.15 (2)	85.19 (4)	85.29 (3)
		NB	76.37 (6)	83.42 (3)	78.78 (5)	82.25 (4)	87.16 (2)	88.27 (1)
		QDA	80.49 (6)	88.52 (3)	84.21 (4)	86.56 (4)	90.29 (2)	92.14 (1)
Avg. Rank of Models			6.00	2.25	4.75	3.50	2.75	1.50

Table 6. FS models Avg. Rank.

S. No.	Datasets	Without FS	FSGA	FSPSO	FSDE	FSACO	FSGJO
1	PC1	6	3	5	3	3	1
2	PC2	5.75	2.25	4.25	3.75	3.25	1.25
3	PC3	6	2.50	5	3.25	3.25	1
4	PC4	5.50	2	4.25	2.50	3	1.75
5	PC5	5.75	2.50	4.75	3.25	3	1
6	JM1	5.50	2.75	4.50	3	1.50	1.25
7	KC1	5.50	2.50	4.25	3.75	2.50	1
8	KC3	6	3.25	4.75	2.75	3	1.25
9	CM1	5.75	2.50	4.75	3.25	2.75	1
10	MC1	4.25	3.25	4.50	3.00	2.75	1.25
11	MC2	5.25	2.00	4.25	3.00	2.50	1.25
12	MW1	6.00	2.25	4.75	3.50	2.75	1.50
Avg. Rank Datasets		5.60	2.56	4.58	3.17	2.77	1.21
		AR6	AR2	AR5	AR4	AR3	AR1

The computation of X_F^2 in Equation (35) involves utilizing the *AvgRankModels*, which is determined to be 15.93. The variables M and N are used to represent the number of datasets and models in the experiment, respectively. The resulting value for the Friedman statistic, F_F, is calculated as 3.98 based on Equation (36). The analysis in this instance is conducted using 12 datasets and 6 models. The critical value is calculated as 2.449 with $(6-1)$ and $(6-1)\times(12-1)$ degrees of freedom, and the significance level of α is 0.05.

$$X_F^2 = \frac{12 \times M}{N \times (N+1)} \times \left[\sum_j AR^2 - \frac{N \times (N+1)^2}{4} \right] \tag{35}$$

$$F_F = \frac{(M-1) \times X_F^2}{M \times (N-1) - X_F^2} \tag{36}$$

The density plot in Figure 9, with a degree of freedom of (5,55), shows the critical value of 2.269. The fact that the Friedman Statistics (F_F = 3.98) is higher than the critical value allows us to reject the null hypothesis (H_0). This implies that there is a noteworthy difference between at least two models.

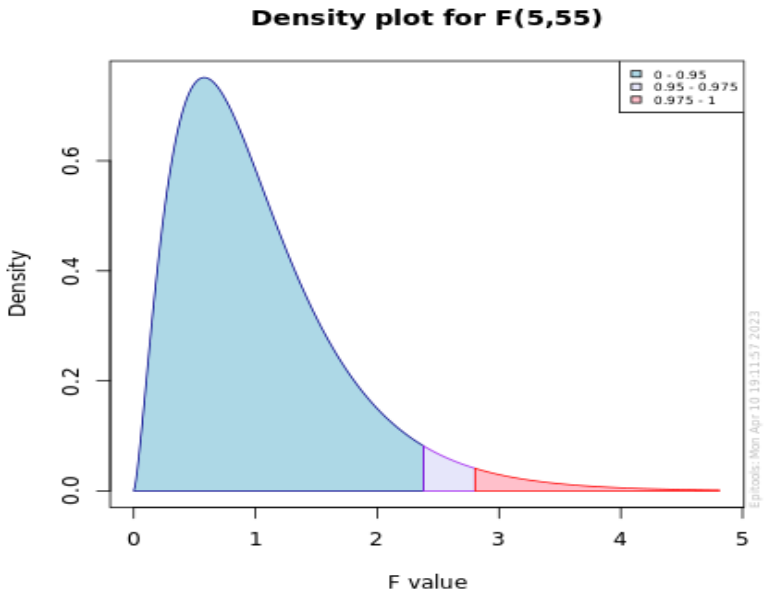

Figure 9. Density plot.

Upon rejecting the null hypothesis in the Friedman test, which implies the presence of variations among multiple models, it is standard practice to conduct a post hoc test to identify the specific models that exhibit significant differences. There are several post hoc tests that can be used, such as the Wilcoxon signed-rank test, the Holm–Bonferroni method, and the Nemenyi test. The choice of the post hoc test depends on the specific research question and the characteristics of the data. The main goal of the post hoc test is to provide more detailed information about the differences between the models and to identify which models are significantly better than others. Additionally, it has been shown to have more statistical power than other methods while maintaining an acceptable type I error rate. Controlling the Type I error rate is important in statistical inference because it helps ensure that the results drawn from the data are accurate and reliable. The Holm procedure [52–56] is a multiple comparison procedure that can be used as a post hoc test after conducting a statistical test like the Friedman test. It uses the *p*-value and z-value to evaluate the performance of each individual. The calculation of z is performed using Equation (37), and then the corresponding *p*-value is obtained from the normal distribution table.

$$z = \frac{AR_x - AR_y}{\sqrt{\frac{N \times (N+1)}{6 \times M}}} \tag{37}$$

The value of z in this context refers to the z-score value, which is calculated using a formula represented by Equation (37). In the formula, N and M represent the quantity of models and datasets used in the experiment, respectively. The average rank of the x^{th} and y^{th} models is symbolized as AR_x and AR_y respectively. Table 7 shows the comparison of all the models using the z-value, *p*-value, and (α/N-i). The significance level used in the assessment is 0.05, denoted by α.

Table 7. Test results of Holm method.

Holm Test				
Sr. No.	**FS Models**	**z Value**	***p* Value**	**Alpha/v-i**
1	FSGJO:WFS	5.755497	0.00001	0.01
2	FSGJO:FSGA	1.77302	0.038114	0.0125
3	FSGJO:FSPSO	4.418912	0.00001	0.016667
4	FSGJO:DE	2.56406	0.005174	0.025
5	FSGJO:ACO	2.045793	0.020393	0.05

Table 7 displays the outcomes of a Holm test carried out on five FS (Feature Selection) models: FSGJO:WFS, FSGJO:FSGA, FSGJO:FSPSO, FSGJO:DE, and FSGJO:ACO. The table presents the alpha/v-I, the *p*-value, and the z-value for each model. The adjusted alpha level (α/N-i) was calculated based on their ranks and the number of models. The adjusted alpha level is the significance level adjusted for the multiple comparisons in the test. It is used to determine if a result is statistically significant after adjusting for the number of tests conducted. The table indicates that, for the most part, the *p*-values are lower than or equal to the adjusted alpha level (α/N-i), except for the FSGJO and FSGA models. These findings indicate that, with the exception of the FSGA model, the FSGJO model exhibits superior and statistically significant results compared to the other models. According to the table, the FSGJO model has noteworthy outcomes and performs better than the other models, except for the FSGA model. There is no statistical significance in the differences of performance among these models.

7. Conclusions

In this study, a novel method for feature selection called FSGJO is introduced, which employs metaheuristic optimization using the GJO algorithm to efficiently identify the optimal set of features. The FSGJO feature selection technique strives to choose the most

significant features within the solution space, aiming to exclude redundant and irrelevant ones. This research assesses the efficiency of the FSGJO method on 12 different datasets using four classifiers (DT, KNN, NB, and QDA). The primary objective is to compare FSGJO's performance with that of existing feature selection models such as FSPSO, FSGA, FSDE, and FSACO, which have different benchmark dimensions. Statistical analysis using the Friedman test indicated that at least two models differed significantly from one another, and the null hypothesis was rejected, which led to the Holm test. Based on the results, it was found that FSGJO displayed a better performance compared to other methods for selecting features, both in relation to accurately classifying data and efficiently eliminating features that were not useful. The advantage of GJO is its ability to handle high-dimensional optimization problems with multiple objectives and avoid local optima by combining exploratory and exploitative search strategies. The only limitation of the proposed model is that its parameters must be adjusted according to the given problem. The proposed method can be applied to other fields such as medical data and gene data.

Author Contributions: Conceptualization, H.D. and S.P.; Methodology, H.D.; Software, S.P.; Validation, M.K.G.; Formal analysis, M.K.G.; Resources, A.A.; Visualization, R.M.P. and M.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data used for this experiment are available in public repository (NASA software defects datasets from PROMISE). The detailed information about the data are provided in the result analysis section.

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

References

1. Catal, C. Software fault prediction: A literature review and current trends. *Expert Syst. Appl.* **2011**, *38*, 4626–4636. [CrossRef]
2. Kundu, A.; Dutta, P.; Ranjit, K.; Bidyadhar, S.; Gourisaria, M.K.; Das, H. Software Fault Prediction Using Machine Learning Models. In Proceedings of the 2022 OITS International Conference on Information Technology (OCIT), Bhubaneswar, India, 14–16 December 2022; IEEE: Manhattan, NY, USA, 2022; pp. 170–175.
3. Malhotra, R. Comparative analysis of statistical and machine learning methods for predicting faulty modules. *Appl. Soft Comput.* **2014**, *21*, 286–297. [CrossRef]
4. Gm, H.; Gourisaria, M.K.; Pandey, M.; Rautaray, S.S. A comprehensive survey and analysis of generative models in machine learning. *Comput. Sci. Rev.* **2020**, *38*, 100285. [CrossRef]
5. Rathore, S.S.; Kumar, S. A decision tree logic based recommendation system to select software fault prediction techniques. *Computing* **2017**, *99*, 255–285. [CrossRef]
6. Rathore, S.S.; Kumar, S. A Decision Tree Regression based Approach for the Number of Software Faults Prediction. *ACM SIGSOFT Softw. Eng. Notes* **2016**, *41*, 1–6. [CrossRef]
7. Singh, Y.; Kaur, A.; Malhotra, R. Software fault proneness prediction using support vector machines. In Proceedings of the World Congress on Engineering, London, UK, 1–3 July 2009; Volume 1, pp. 1–3.
8. Li, J.; He, P.; Zhu, J.; Lyu, M.R. Software defect prediction via convolutional neural network. In Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, 25–29 July 2017; IEEE: Manhattan, NY, USA, 2017; pp. 318–328.
9. Goyal, J.; Ranjan Sinha, R. Software defect-based prediction using logistic regression: Review and challenges. In Proceedings of the Second International Conference on Sustainable Technologies for Computational Intelligence: Proceedings of ICTSCI 2021, Dehradun, India, 22–23 May 2021; Springer: Singapore, 2022; pp. 233–248.
10. Chandrashekar, G.; Sahin, F. A survey on feature selection methods. *Comput. Electr. Eng.* **2014**, *40*, 16–28. [CrossRef]
11. Brezočnik, L.; Fister, I., Jr.; Podgorelec, V. Swarm intelligence algorithms for feature selection: A review. *Appl. Sci.* **2018**, *8*, 1521. [CrossRef]
12. Cherrington, M.; Thabtah, F.; Lu, J.; Xu, Q. Feature Selection: Filter Methods Performance Challenges. In Proceedings of the 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 3–4 April 2019; pp. 1–4. [CrossRef]
13. Gayatri, N.; Nickolas, S.; Reddy, A.V. ANOVA discriminant analysis for features selected through decision tree induction method. In *Global Trends in Computing and Communication Systems*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 61–70.
14. Chen, G.; Chen, J. A novel wrapper method for feature selection and its applications. *Neurocomputing* **2015**, *159*, 219–226. [CrossRef]

15. Zeng, X.; Chen, Y.W.; Tao, C. Feature selection using recursive feature elimination for handwritten digit recognition. In Proceedings of the 2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, Kyoto, Japan, 12–14 September 2009; IEEE: Manhattan, NY, USA, 2009; pp. 1205–1208.
16. Borboudakis, G.; Tsamardinos, I. Forward-backward selection with early dropping. *J. Mach. Learn. Res.* **2019**, *20*, 276–314.
17. Lal, T.N.; Chapelle, O.; Weston, J.; Elisseeff, A. Embedded methods. In *Feature Extraction: Foundations and Applications*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 137–165.
18. Chen, C.-W.; Tsai, Y.-H.; Chang, F.-R.; Lin, W.-C. Ensemble feature selection in medical datasets: Combining filter, wrapper, and embedded feature selection results. *Expert Syst.* **2020**, *37*, e12553. [CrossRef]
19. Muthukrishnan, R.; Rohini, R. LASSO: A feature selection technique in predictive modeling for machine learning. In Proceedings of the 2016 IEEE International Conference on Advances in Computer Applications (ICACA), Coimbatore, India, 24 October 2016; pp. 18–20. [CrossRef]
20. Paul, S.; Drineas, P. Feature Selection for Ridge Regression with Provable Guarantees. *Neural Comput.* **2016**, *28*, 716–742. [CrossRef] [PubMed]
21. Song, F.; Guo, Z.; Mei, D. Feature selection using principal component analysis. In Proceedings of the 2010 International Conference on System Science, Engineering Design and Manufacturing Informatization, Yichang, China, 12–14 November 2010; IEEE: Manhattan, NY, USA, 2010; Volume 1, pp. 27–30.
22. Belkina, A.C.; Ciccolella, C.O.; Anno, R.; Halpert, R.; Spidlen, J.; Snyder-Cappione, J.E. Automated optimized parameters for T-distributed stochastic neighbor embedding improve visualization and analysis of large datasets. *Nat. Commun.* **2019**, *10*, 5415. [CrossRef] [PubMed]
23. Malhotra, R.; Pritam, N.; Singh, Y. On the applicability of evolutionary computation for software defect prediction. In Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 24–27 September 2014; pp. 2249–2257. [CrossRef]
24. Ab Wahab, M.N.; Nefti-Meziani, S.; Atyabi, A. A Comprehensive Review of Swarm Optimization Algorithms. *PLoS ONE* **2015**, *10*, e0122827. [CrossRef] [PubMed]
25. Prajapati, S.; Das, H.; Gourisaria, M.K. Feature selection using genetic algorithm for microarray data classification. In Proceedings of the 2022 OPJU International Technology Conference on Emerging Technologies for Sustainable Development (OTCON), Raigarh, India, 8–10 February 2022.
26. Du, K.L.; Swamy MN, S.; Du, K.L.; Swamy, M.N.S. Particle swarm optimization. In *Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 153–173.
27. Brezočnik, L.; Podgorelec, V. Applying weighted particle swarm optimization to imbalanced data in software defect prediction. In *New Technologies, Development and Application 4*; Springer International Publishing: Manhattan, NY, USA, 2019; pp. 289–296.
28. Das, S.; Suganthan, P.N. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Trans. Evol. Comput.* **2010**, *15*, 4–31. [CrossRef]
29. Prajapati, S.; Das, H.; Gourisaria, M.K. Feature Selection using Ant Colony Optimization for Microarray Data Classification. In Proceedings of the 2023 6th International Conference on Information Systems and Computer Networks (ISCON), Mathura, India, 3–4 March 2023; pp. 1–6.
30. Chopra, N.; Ansari, M.M. Golden jackal optimization: A novel nature-inspired optimizer for engineering applications. *Expert Syst. Appl.* **2022**, *198*, 116924. [CrossRef]
31. Prasad, D. *Automated Software Testing: Foundations Applications Challenges*; Jena, A.K., Das, H., Mohapatra, D.P., Eds.; Springer: Berlin/Heidelberg, Germany, 2020; pp. 1–165.
32. Das, H.; Gourisaria, M.K.; Sah, B.K.; Bilgaiyan, S.; Badajena, J.C.; Pattanayak, R.M. E-Healthcare System for Disease Detection Based on Medical Image Classification Using, C.N.N. In *Empirical Research for Futuristic E-Commerce Systems: Foundations and Applications*; IGI Global: Hershey, PA, USA; pp. 213–230.
33. Prajapati, S.; Das, H.; Gourisaria, M.K. Microarray data classification using machine learning algorithms. In Proceedings of the 2022 OPJU International Technology Conference on Emerging Technologies for Sustainable Development (OTCON), Raigarh, India, 8–10 February 2022.
34. Das, H.; Naik, B.; Behera, H.; Jaiswal, S.; Mahato, P.; Rout, M. Biomedical data analysis using neuro-fuzzy model with post-feature reduction. *J. King Saud Univ. Comput. Inf. Sci.* **2020**, *34*, 2540–2550. [CrossRef]
35. Das, H.; Naik, B.; Behera, H.S. Medical disease analysis using neuro-fuzzy with feature extraction model for classification. *Inform. Med. Unlocked* **2020**, *18*, 100288. [CrossRef]
36. Das, H.; Naik, B.; Behera, H.S. An experimental analysis of machine learning classification algorithms on biomedical data. In Proceedings of the 2nd International Conference on Communication, Devices and Computing, Haldia, India, 14–15 March 2020; Springer: Singapore, 2020; pp. 525–539.
37. Saha, I.; Gourisaria, M.K.; Harshvardhan, G.M. Classification System for Prediction of Chronic Kidney Disease Using Data Mining Techniques. In *Advances in Data and Information Sciences: Proceedings of ICDIS 2021*; Springer: Singapore, 2022; pp. 429–443.
38. Agarwal, S.; Tomar, D. A Feature Selection Based Model for Software Defect Prediction. *Int. J. Adv. Sci. Technol.* **2014**, *65*, 39–58. [CrossRef]
39. Turabieh, H.; Mafarja, M.; Li, X. Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. *Expert Syst. Appl.* **2019**, *122*, 27–42. [CrossRef]

40. Zorarpacı, E.; Özel, S.A. A hybrid approach of differential evolution and artificial bee colony for feature selection. *Expert Syst. Appl.* **2016**, *62*, 91–103. [CrossRef]
41. Ibrahim, D.R.; Ghnemat, R.; Hudaib, A. Software defect prediction using feature selection and random forest algorithm. In Proceedings of the 2017 International Conference on New Trends in Computing Sciences (ICTCS), Amman, Jordan, 11–13 October 2017; IEEE: Manhattan, NY, USA, 2017; pp. 252–257.
42. Wahono, R.S.; Suryana, N. Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction. *Int. J. Softw. Eng. Its Appl.* **2013**, *7*, 153–166. [CrossRef]
43. Rong, X.; Li, F.; Cui, Z. A model for software defect prediction using support vector machine based on CBA. *Int. J. Intell. Syst. Technol. Appl.* **2016**, *15*, 19–34. [CrossRef]
44. Wahono, R.S.; Suryana, N.; Ahmad, S. Metaheuristic Optimization based Feature Selection for Software Defect Prediction. *J. Softw.* **2014**, *9*, 1324–1333. [CrossRef]
45. Jacob, S.G. Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques. *Int. J. Comput. Appl.* **2015**, *117*, 18–22.
46. Das, H.; Naik, B.; Behera, H.S. Optimal Selection of Features Using Artificial Electric Field Algorithm for Classification. *Arab. J. Sci. Eng.* **2021**, *46*, 8355–8369. [CrossRef]
47. Das, H.; Naik, B.; Behera, H. A Jaya algorithm based wrapper method for optimal feature selection in supervised classification. *J. King Saud Univ. Comput. Inf. Sci.* **2020**, *34*, 3851–3863. [CrossRef]
48. Padhi, B.K.; Chakravarty, S.; Naik, B.; Pattanayak, R.M.; Das, H. RHSOFS: Feature Selection Using the Rock Hyrax Swarm Optimization Algorithm for Credit Card Fraud Detection System. *Sensors* **2022**, *22*, 9321. [CrossRef]
49. Dutta, H.; Gourisaria, M.K.; Das, H. Wrapper Based Feature Selection Approach Using Black Widow Optimization Algorithm for Data Classification. In *Computational Intelligence in Pattern Recognition: Proceedings of CIPR 2022*; Springer Nature: Singapore, 2022; pp. 487–496.
50. Anbu, M.; Mala, G.S.A. Feature selection using firefly algorithm in software defect prediction. *Clust. Comput.* **2019**, *22*, 10925–10934. [CrossRef]
51. Demšar, J. Statistical comparisons of classifers over multiple data sets. *J. Mach. Learn. Res.* **2006**, *7*, 1–30.
52. Friedman, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Am. Stat. Assoc.* **1937**, *32*, 675–701. [CrossRef]
53. Friedman, M. A comparison of alternative tests of signifcance for the problem of m rankings. *Ann. Math. Stat.* **1940**, *11*, 86–92. [CrossRef]
54. Iman, R.L.; Davenport, J.M. Approximations of the critical region of the fbietkan statistic. *Commun. Stat. Theory Methods* **1980**, *9*, 571–595. [CrossRef]
55. García, S.; Fernández, A.; Luengo, J.; Herrera, F. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Inf. Sci.* **2010**, *180*, 2044–2064. [CrossRef]
56. Luengo, J.; García, S.; Herrera, F. A study on the use of statistical tests for experimentation with neural networks: Analysis of parametric test conditions and non-parametric tests. *Expert Syst. Appl.* **2009**, *36*, 7798–7808. [CrossRef]