A project report on

NATURE INSPIRED APPROACHES IN SOFTWARE FAULT PREDICTION

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology (IT)

Under the supervision of

Mr. Sachin Garg
Assistant Professor

Mr. Varun Goel
Assistant Professor

Submitted by

Tushar Arora
(20514803119)

Harshit Saini
(20214803119)

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

SECTOR-22, ROHINI, DELHI –110086

June 2023

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Mr. Sachin Garg (Assistant Professor) and Mr. Varun Goel (Assistant Professor),** Department of Information Technology, MAIT, for their guidance with unsurpassed knowledge and immense encouragement.

We are grateful to **Dr. M.L. Sharma**, Head of the Department, Information Technology, for providing us with the required facilities for the completion of the project work.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

Tushar Arora
20514803119

Harshit Saini
20214803119

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this project report titled, **"NATURE INSPIRED APPROACHES IN SOFTWARE FAULT PREDICTION"** submitted by me in the partial fulfillment of the requirement of the award of the degree of **Bachelor of Technology (B.Tech.)** Submitted in the Department of **Information Technology,** Maharaja Agrasen Institute of Technology is an authentic record of my project work carried out under the guidance of Mr. Varun Goel and Mr. Sachin Garg. The matter presented in this project report has not been submitted either in part or full to any university or Institute for award of any degree.

Date :	**Tushar Arora**	**Harshit Saini**
Place: Delhi	Roll No.: 20514803119	Roll No.: 20214803119

SUPERVISOR'S CERTIFICATE

It is to certify that the Project entitled **"NATURE INSPIRED APPROACHES IN SOFTWARE FAULT PREDICTION"** which is being submitted by **Mr. Sachin Garg and Mr. Varun Goel** to the Maharaja Agrasen Institute of Technology, Rohini in the fulfillment of the requirement for the award of the degree of **Bachelor of Technology (B.Tech.)**, is a record of bonafide project work carried out by him/her under my/ our guidance and supervision.

Mr. Sachin Garg
Assistant Professor
Department of IT

Mr. Varun Goel
Assistant Professor
Department of IT

Prof. (Dr.) M.L. Sharma
H.O.D.
Department of IT

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In recent years, the use of nature-inspired approaches in software fault prediction has gained significant attention within the field of software engineering. This paper aims to explore the effectiveness of six nature-inspired algorithms, namely Ant Colony, Particle Swarm Optimization, Firefly, Bat, Harris Hawks, and Genetic Algorithm, in predicting software faults. To evaluate the algorithms, three widely used datasets, namely JM1, CM1, and PC1, are employed. Through extensive experimentation and analysis, our study reveals promising results regarding the capability of nature-inspired approaches to predict software faults. The outcomes demonstrate that these algorithms can effectively anticipate software faults, although their performance varies depending on the dataset employed. Certain algorithms exhibit superior performance when compared to others, emphasizing the significance of selecting appropriate techniques based on the dataset characteristics. The findings of this research suggest that nature-inspired approaches hold substantial potential as practical and efficient methods for software fault prediction. Leveraging the inherent wisdom of natural systems, such as the collective intelligence of ant colonies or the adaptive behaviors of genetic algorithms, can contribute to enhancing software quality and minimizing costs. By embracing these innovative approaches, software engineers can enhance fault prediction mechanisms, ultimately leading to more reliable and robust software systems.

CHAPTER 1: INTRODUCTION

1.1 Need of the study

The IT and software industry has grown tremendously over the past few years, creating an increasing impact on the lives of people and on society as a whole. Consequently, we must make the software and applications more accurate, free of major errors, and more reliable. Therefore, predicting software flaws could be very useful in the IT field and will have a profound impact on society at large.

We can predict system faults and bugs by analyzing software systems, which include databases and local state machines. We take a very methodical approach to finding bugs, optimizing the program for efficiency and quality. This may lead to improved customer satisfaction, increased customer retention and save money.

1.2 Scope of the study

Machine learning techniques have been around us and have been compared and used for analysis for many kinds of data science applications. The major motivation behind this research-based project was to explore the feature selection methods, data preparation, and processing behind the training models in machine learning. With first- hand models and libraries, the challenge we face today is data were beside their abundance, and in our cooked models, the accuracy we see during training, testing, and actual validation has a higher variance. Hence this project is carried out with the motivation to explore behind the models, and further implement Logistic Regression model to train the obtained data.

Software fault prediction aims to predict fault-prone software modules by using some under- lying properties of the software project. It is typically performed by training a prediction model using project properties augmented with fault information for a known project, and subsequently using the prediction model to predict faults for unknown projects. Software fault prediction is based on the understanding that if a project developed in an environment leads to faults, then any module developed in the similar environment with similar project characteristics will ends to be faulty. The early detection of faulty modules can be useful to streamline the efforts to be applied in the later phases of software development by better focusing quality assurance efforts to those modules.

1.3 Objectives of the study

The main objectives of developing this project are:

- Evaluate the effectiveness of six nature-inspired algorithms (Ant Colony, Particle Swarm Optimization, Firefly, Bat, Harris Hawks, and Genetic Algorithm) in predicting software faults.
- Compare the performance of these nature-inspired algorithms using three commonly used datasets (JM1, CM1, and PC1) in order to assess their suitability for different types of software systems.
- Investigate the potential of nature-inspired approaches as practical and efficient means for software fault prediction.
- Identify which nature-inspired algorithm(s) demonstrate superior performance in predicting software faults and understand the factors that contribute to their effectiveness.
- Provide empirical evidence and insights into the applicability of nature-inspired approaches in software fault prediction, contributing to the existing body of knowledge in the field of software engineering.
- Enable software engineers to make informed decisions regarding the selection and utilization of nature-inspired algorithms for effective software fault prediction.
- Contribute to improving software quality and reducing costs by developing reliable and robust fault prediction mechanisms using nature-inspired approaches.

CHAPTER 2: LITERATURE REVIEW

Software fault prediction is a critical task in software engineering, aiming to identify potential faults in software systems before they occur. Over the years, researchers have explored various techniques and methodologies to enhance the accuracy and efficiency of software fault prediction. In recent times, nature-inspired approaches have gained considerable attention due to their ability to leverage natural phenomena and principles for solving complex problems. This literature review presents an overview of the existing research on the use of nature-inspired algorithms in software fault prediction and highlights their effectiveness and potential contributions to the field.

Nature-inspired algorithms draw inspiration from natural systems and phenomena, such as ant colonies, particle swarms, fireflies, bats, hawks, and genetic evolution. These algorithms mimic the behavior and adaptive characteristics of these natural systems to solve complex optimization and prediction problems. In the context of software fault prediction, these algorithms offer the potential to overcome the limitations of traditional techniques and provide more accurate and efficient predictions.

Several studies have investigated the effectiveness of nature-inspired algorithms in software fault prediction. To predict and prevent bugs in production researchers have implemented and worked on various machine-learning approaches. It is known that software maintenance is the most expensive phase in the software development lifecycle [4]. A software defect predictive model enables organizations to help to reduce the maintenance effort, time and cost overall on a software project [3] [4]. The various researched algorithms are a result of various findings and correlations between some software metrics and fault proneness [11].This paper uses 4 of many ML classifiers as suggested by a recent systematic literature study [4]. As two studies stated that machine learning-based models for software fault prediction [11] [12].

The application of nature-inspired approaches in software fault prediction offers several potential contributions. These approaches can enhance software quality by identifying and addressing faults in the early stages of development. By optimizing the allocation of testing resources, nature-inspired algorithms can significantly reduce costs associated with fault detection and correction. Additionally, the adaptive nature of these algorithms allows for continuous learning and improvement, making them suitable for dynamic software environments.

CHAPTER 3: IMPLEMENTATION OF PROPOSED SYSTEM

3.1 System Requirements

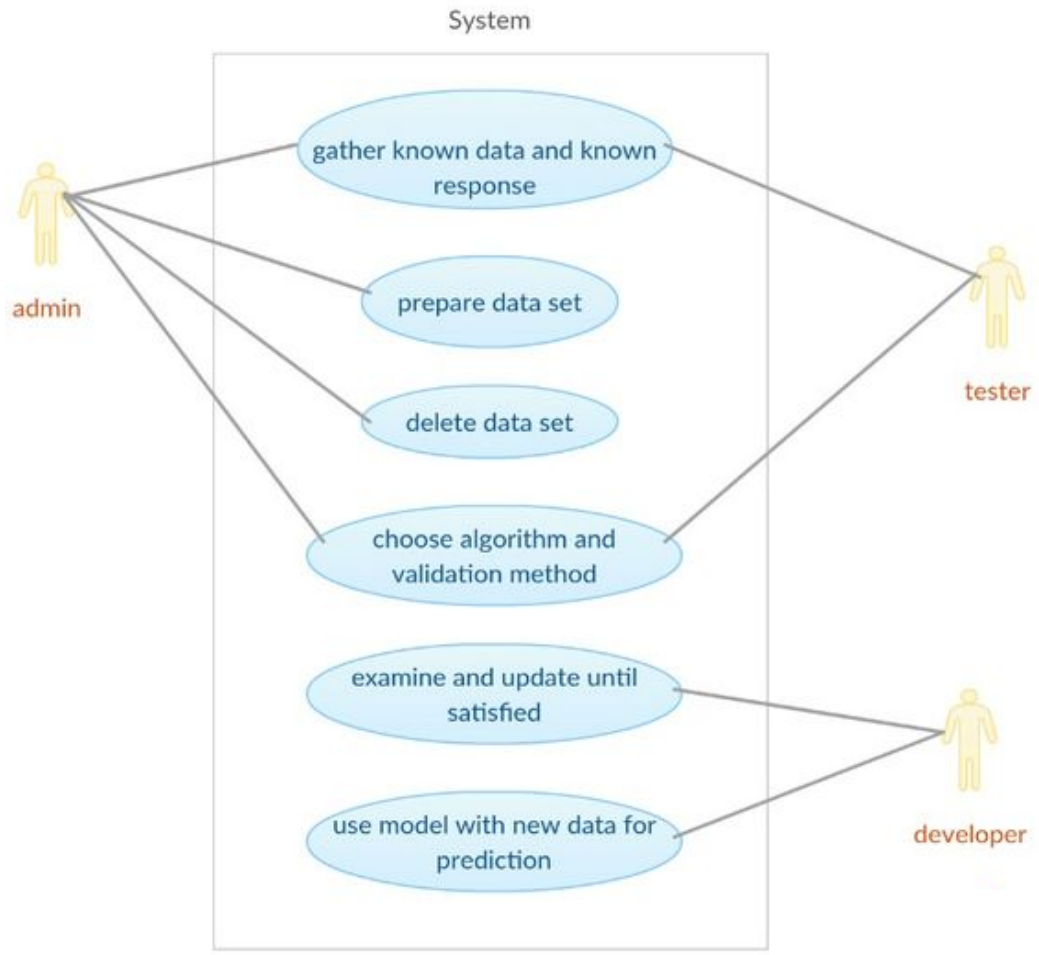
3.1.1 Hardware requirements:

Processer	:	Any Update Processer
Ram	:	Min 4GB
Hard Disk	:	100GB Free space

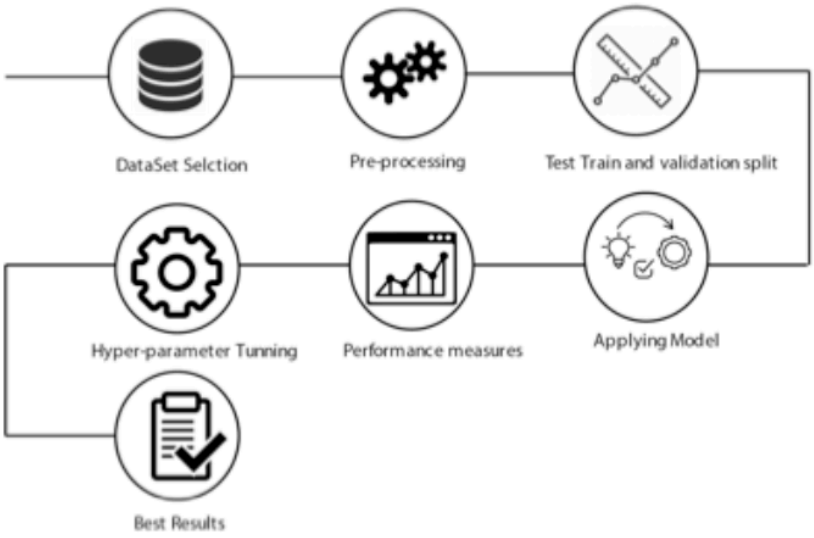
3.1.2 Software requirements:

Operating System	:	Windows Family Technology (Python3.7)
IDE	:	Jupyter notebook

3.2 Use Case Diagram



3.3 Flow Chart



3.4 Proposed System

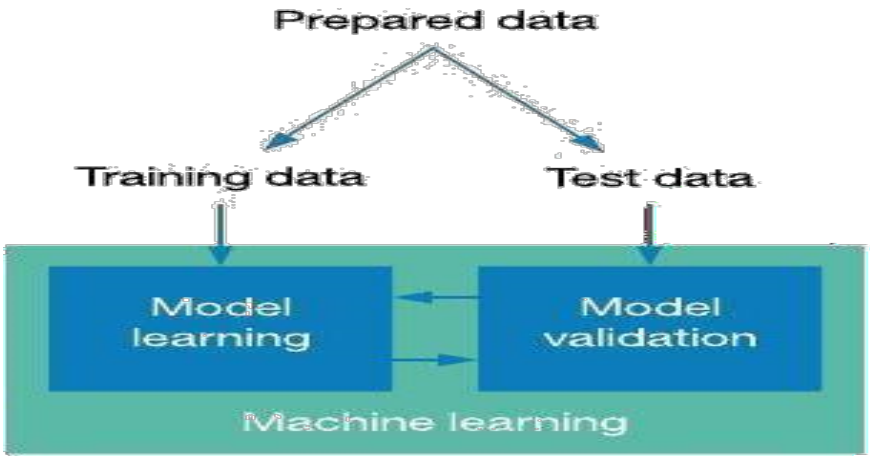
The working of the system starts with the collection of data and selecting the important attributes. Then the required data is preprocessed into the required format. The data is then divided into two parts training and testing data. The algorithms are applied and the model is trained using the training data. The accuracy of the system is obtained by testing the system using the testing data. This system is implemented using the following modules.

1. Collection of Dataset
2. Selection of attributes
3. Data Pre-Processing
4. Balancing of Data
5. Fault Prediction

3.4.1 Collection of Dataset

Initially, we collect a dataset for our software fault prediction system. After the collection of the dataset, we split the dataset into training data and testing data. The training dataset is used for prediction model learning and testing data is used for evaluating the prediction model.

For this project, 70% of training data is used and 30% of data is used for testing. The dataset used for this project is JM1 an CM1. The dataset consists of 22 attributes and we are using all of them.



3.4.2 Selection of attributes

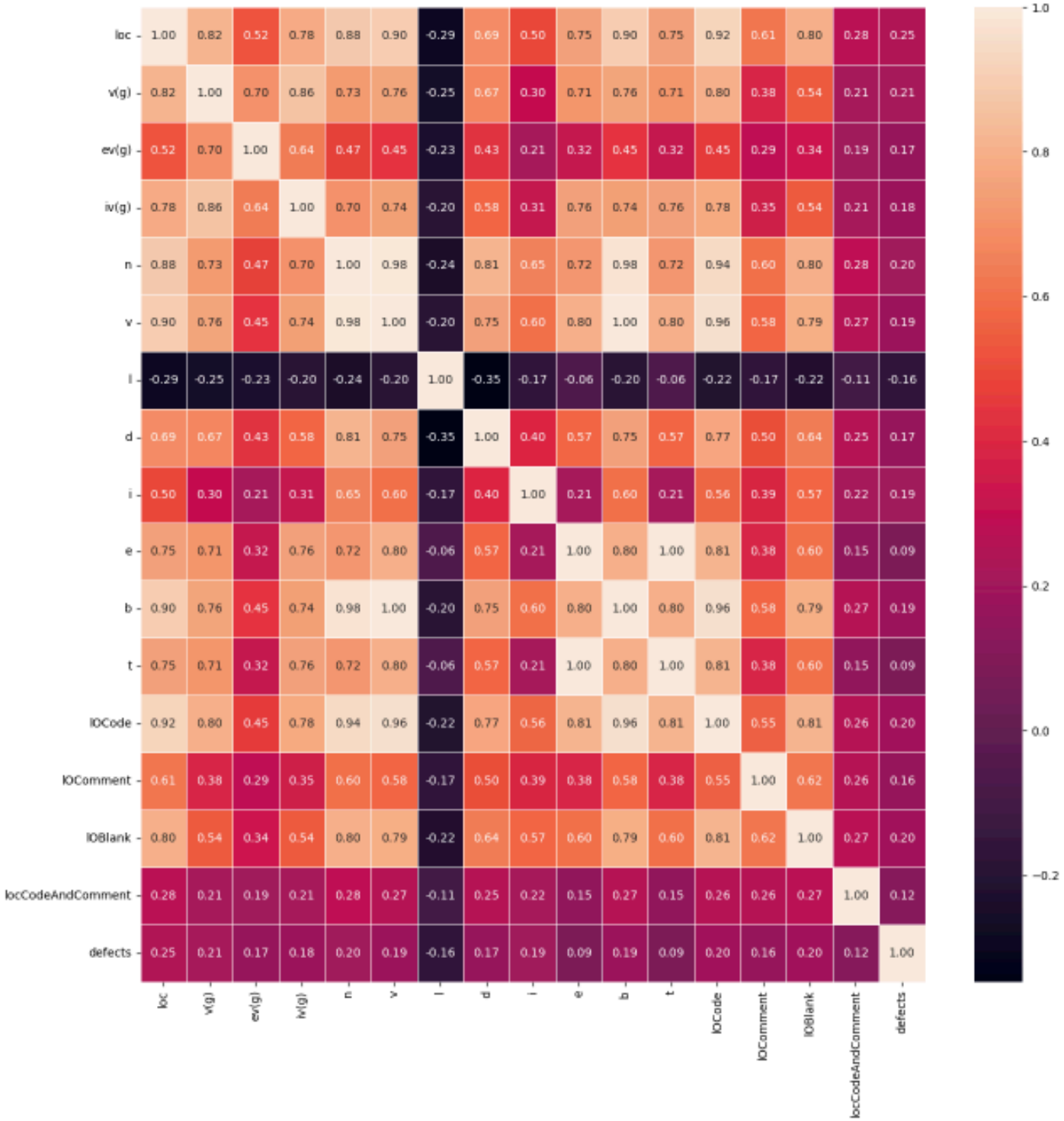
Attribute or Feature selection includes the selection of appropriate attributes for the prediction system. This is used to increase the efficiency of the system. Various attributes of the software like loc, branch count, defects, unique operands, etc. are selected for the prediction. The Correlation matrix is used for attribute selection for this model.

3.4.3 Data Pre-Processing

Data pre-processing is an important step for the creation of a machine learning model. Initially, data may not be clean or in the required format for the model which can cause misleading outcomes. In pre-processing of data, we transform data into our required format. It is used to deal with noises, duplicates, and missing values of the dataset. Data pre-processing has the activities like importing datasets, splitting datasets, attribute scaling, etc. Preprocessing of data is required for improving the efficiency and accuracy of the model / system.

3.4.4 Balancing of Data

Imbalanced datasets can be balanced in two ways. They are Under Sampling and Over Sampling (a) (a) Under Sampling: In Under Sampling, dataset balance is done by the reduction of the size of the ample class. This process is considered when the amount of data is adequate.



(b) Over Sampling: In Over Sampling, dataset balance is done by increasing the size of the scarce samples. This process is considered when the amount of data is inadequate.

3.4.5 Fault Prediction

Various nature inspired algorithms like Ant Colony, Particle Swarm Optimization, Firefly, Bat, Harris Hawks, etc are used for optimization and prediction. Comparative analysis is performed among algorithms and the algorithm that gives the highest accuracy is used for software fault prediction.

CHAPTER 4: EXPERIMENTAL ANALYSIS

4.1 Machine Learning

In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.

- **Supervised Learning**

Supervised learning is the type of machine learning in which machines are trained using well "labeled" training data, and on the basis of that data, machines predict the output. The labeled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to find a mapping function to map the input variable(x) with the output variable(y).

- **Unsupervised learning**

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar to how a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

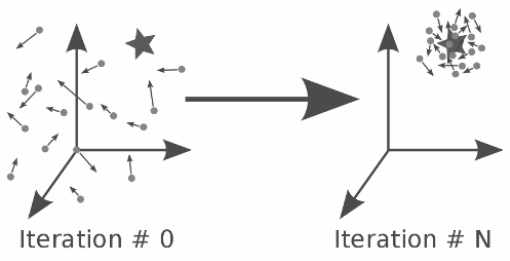
- **Reinforcement learning**

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

4.2 Nature Inspired Algorithms

4.2.1 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a nature-inspired optimization algorithm that mimics the collective behavior of bird flocks or fish schools to solve complex optimization problems. It was first introduced by Kennedy and Eberhart in 1995. The algorithm's foundation lies in the principles of social interaction and information sharing among individuals within a swarm. This section provides an in-depth understanding of PSO, including its mathematical formulation, advantages, disadvantages, and any relevant diagrams or graphs.



Mathematical Formulation:

PSO operates based on the concept of particles moving through a multidimensional search space to find the optimal solution. Let's consider a swarm of N particles in a D-dimensional search space. Each particle's position is represented as a D-dimensional vector, denoted as X = (x_1, x_2, ..., x_D). The particle's velocity is also represented as a D-dimensional vector, V = (v_1, v_2, ..., v_D). The objective is to optimize a fitness function f(X) that evaluates the quality of a particle's position.

At each iteration, the particle updates its position and velocity based on its own experience and the experiences of its neighboring particles. The velocity update equation is as follows:

$V(t+1) = \omega * V(t) + c_1 * r_1 * (Pbest - X(t)) + c_2 * r_2 * (Gbest - X(t))$

where V(t+1) represents the updated velocity, V(t) is the current velocity, X(t) denotes the current position, Pbest is the best position achieved by the particle so far, Gbest represents the best position achieved by any particle in the swarm, ω is the inertia weight, c_1 and c_2 are acceleration coefficients, and r_1 and r_2 are random numbers between 0 and 1.

The position update equation is given by:

$X(t+1) = X(t) + V(t+1)$

The process continues iteratively until a termination condition is met, such as reaching a maximum number of iterations or achieving a desired fitness value.

Advantages of PSO:

- Simplicity: PSO is relatively easy to understand and implement compared to other optimization algorithms. Its straightforward concept makes it accessible to researchers and practitioners with different levels of expertise.
- Global Search Capability: PSO is capable of performing a global search by exploring different regions of the search space simultaneously. This ability allows it to avoid getting trapped in local optima and increases the likelihood of finding the global optimum.
- Robustness: PSO exhibits robustness against noisy and dynamic environments. The swarm's ability to adapt and adjust its search trajectory based on individual and collective experiences enables it to handle changing problem landscapes effectively.
- Few Parameters: PSO has few parameters that require tuning, making it less time-consuming and simpler to apply in practice compared to other optimization algorithms.

Disadvantages of PSO:

- Premature Convergence: PSO is prone to premature convergence, where the swarm prematurely converges to a suboptimal solution without exploring the entire search space. This can occur if the swarm lacks diversity or becomes trapped in local optima.

- Lack of Diversity: The exploration capability of PSO heavily relies on the diversity within the swarm. If the particles converge too quickly or get stuck in a narrow region, the algorithm's performance may suffer.

- Parameter Sensitivity: Although PSO has fewer parameters compared to some other algorithms, the performance of PSO can be sensitive to the setting of these parameters. Selecting appropriate values for the acceleration coefficients, inertia weight, and other parameters can significantly impact the algorithm's convergence and exploration abilities.

4.2.2 Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is a metaheuristic algorithm inspired by the foraging behavior of ants. It was introduced by Marco Dorigo in the early 1990s. ACO utilizes the concept of pheromone trails left by ants to find optimal solutions to complex optimization problems. This section provides a comprehensive explanation of ACO, including its mathematical formulation, advantages, disadvantages, and relevant diagrams or graphs.

Mathematical Formulation:

ACO is typically applied to combinatorial optimization problems, such as the Traveling Salesman Problem (TSP). Let's consider a set of N cities that need to be visited. An ant constructively builds a tour by visiting each city exactly once and returning to the starting city. The optimization objective is to minimize the total distance traveled.

The algorithm begins by initializing the pheromone trail values on each edge between cities. The pheromone trail represents the attractiveness of a particular path. At each iteration, each ant probabilistically selects the next city to visit based on a combination of pheromone trail information and a heuristic value, which measures the desirability of moving from the current city to a potential next city. The pheromone update rule is as follows:

$$\tau(i, j) = (1 - \rho) * \tau(i, j) + \sum(\Delta\tau\_k)$$

where $\tau(i, j)$ represents the pheromone value on the edge between cities i and j, ρ is the evaporation rate, and $\Delta\tau\_k$ denotes the amount of pheromone deposited by ant k on the edge.

The ants continue constructing their tours until all cities have been visited. The pheromone values are updated based on the quality of the solutions found by the ants. This process iteratively

continues until a termination criterion is met, such as a maximum number of iterations or convergence to a satisfactory solution.

Advantages of ACO:

- Robustness: ACO is highly robust and capable of finding good solutions even in complex and dynamic problem landscapes. The pheromone trails allow ants to collectively adapt their search behavior based on the information exchanged, enabling effective exploration and exploitation.

- Global Search Capability: ACO excels in performing a global search by exploring various paths simultaneously. The pheromone reinforcement mechanism helps the ants converge toward the optimal solution by intensifying the exploration of promising regions.

- Versatility: ACO can be applied to a wide range of combinatorial optimization problems, including the Traveling Salesman Problem, the Vehicle Routing Problem, and the Quadratic Assignment Problem. Its versatility makes it a valuable tool for solving diverse real-world problems.

- Low Computational Complexity: ACO has a relatively low computational complexity, allowing it to handle large-scale optimization problems efficiently. It is particularly effective for problems where traditional optimization techniques may struggle due to their high computational requirements.

Disadvantages of ACO:

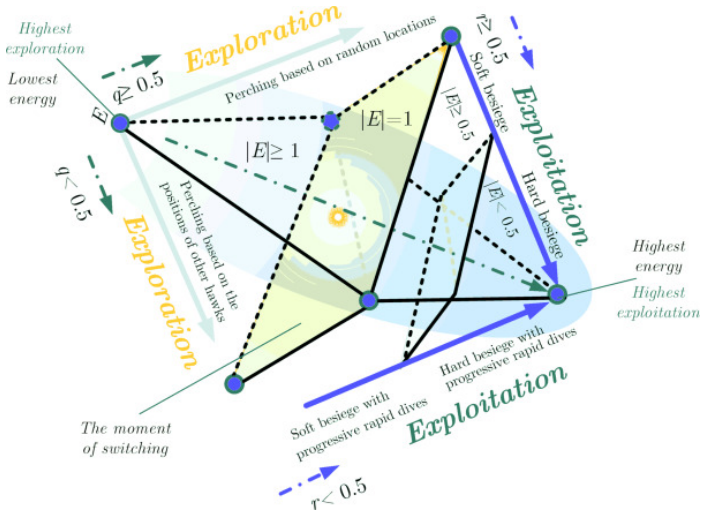
- Premature Convergence: ACO is prone to premature convergence, where the search prematurely focuses on a suboptimal solution due to excessive exploitation of the best path found so far. The algorithm may struggle to escape from local optima, limiting its ability to find the global optimum.

- Parameter Sensitivity: ACO performance can be sensitive to parameter settings, such as the evaporation rate and heuristic information. Fine-tuning these parameters can significantly impact the algorithm's convergence speed and solution quality.

- Scalability: ACO's performance may deteriorate as the problem size increases. The algorithm's reliance on pheromone communication and construction of complete tours can become computationally expensive for larger problem instances.

4.2.3 Harris Hawks Algorithm

The Harris Hawks Algorithm (HHA) is a nature-inspired optimization algorithm inspired by the hunting behavior of Harris's hawks, a species of raptors. This algorithm aims to solve optimization problems by simulating the collaboration and hunting strategies observed in Harris's hawks. However, it's important to note that the Harris Hawks Algorithm may not have as extensive a body of literature or scientific validation as some other well-established optimization algorithms.



Mathematical Formulation:

The specific mathematical formulation of the Harris Hawks Algorithm is not widely documented or published in scientific literature. However, the algorithm is believed to be based on the principles of population-based optimization, where a group of virtual hawks collaboratively search for an optimal solution in a search space. The algorithm likely involves updating the positions of the hawks based on certain rules, such as their individual experiences and interactions with other hawks.

Advantage:

- Collaboration and Collective Intelligence: The Harris Hawks Algorithm is inspired by the collaborative hunting behavior of Harris's hawks, which suggests that the algorithm may benefit from the collective intelligence of the population. This collaboration can potentially enhance the algorithm's exploration and exploitation capabilities in the search space.

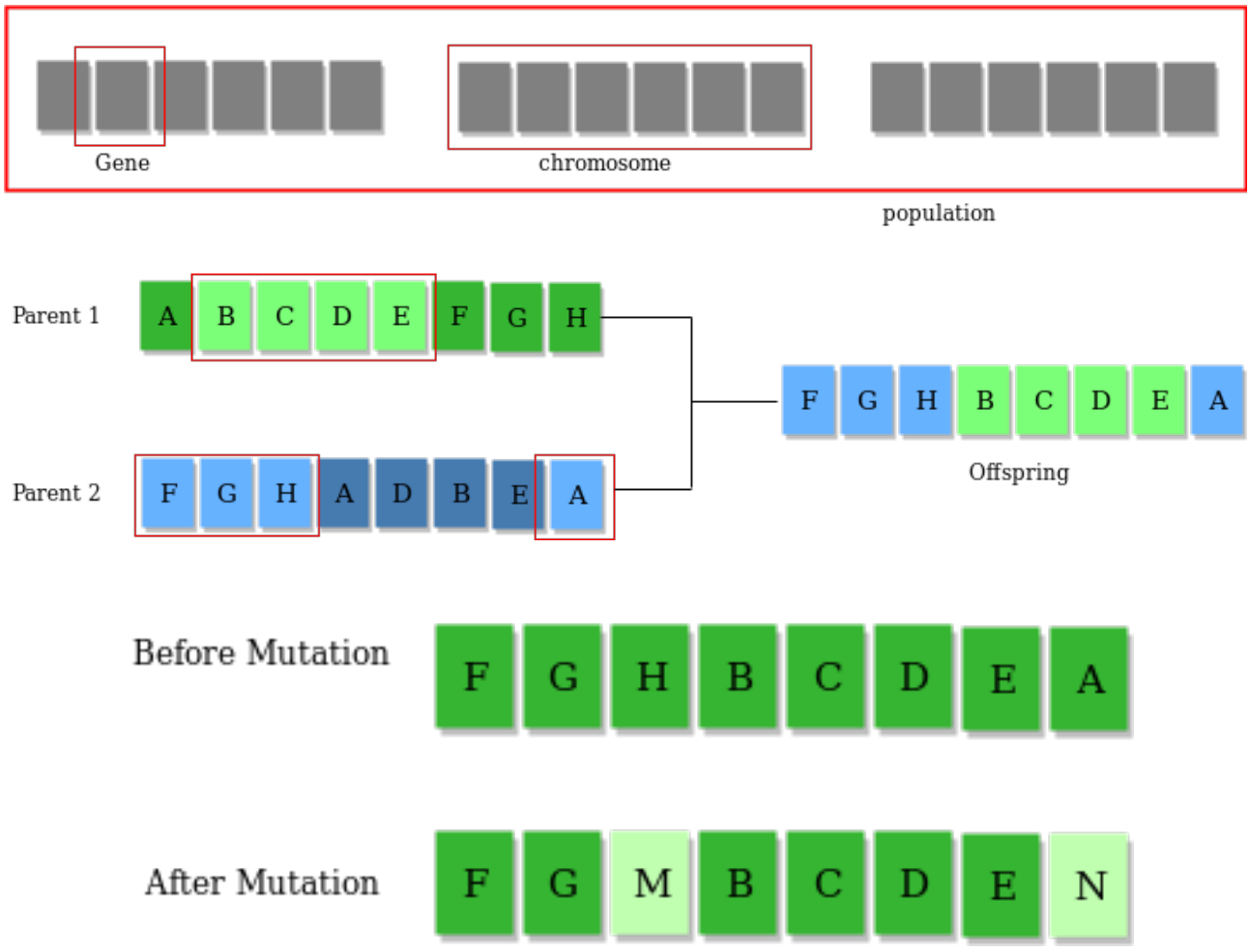
Disadvantage:

- Limited Research and Validation: The Harris Hawks Algorithm is relatively new and may not have undergone extensive research or validation compared to more established optimization

algorithms. As a result, there may be limited scientific literature, case studies, or empirical evidence supporting its effectiveness or efficiency in solving optimization problems.

4.2.4 Genetic Algorithm

Genetic Algorithm (GA) is a widely used optimization technique inspired by the process of natural selection and genetics. It mimics the principles of evolution to search for optimal solutions to complex problems. This section provides a comprehensive explanation of the Genetic Algorithm, including its mathematical formulation, advantages, disadvantages, and relevant diagrams or graphs.



Mathematical Formulation:

The Genetic Algorithm operates on a population of candidate solutions, commonly referred to as chromosomes or individuals. Each chromosome represents a potential solution to the optimization problem. The algorithm evolves the population over generations by applying selection, crossover, and mutation operators.

1. Initialization: An initial population of chromosomes is randomly generated. Each chromosome encodes a potential solution to the problem.

2. Evaluation: The fitness of each chromosome is evaluated using an objective function that quantifies the quality or performance of the solution.

3. Selection: A subset of chromosomes is selected for reproduction based on their fitness values. Higher fitness individuals are more likely to be selected, mimicking the concept of survival of the fittest.

4. Crossover: The selected chromosomes undergo crossover, which involves exchanging genetic material to create new offspring. Crossover mimics genetic recombination in natural reproduction and promotes the exploration of the solution space.

5. Mutation: A small probability of mutation is applied to each offspring. Mutation introduces random changes in the genetic material, allowing the algorithm to explore new areas of the solution space that may not be reachable through selection and crossover.

6. Replacement: The offspring replace a portion of the existing population, typically the least fit individuals. This ensures that the population evolves towards better solutions over generations.

7. Termination: The algorithm terminates when a termination criterion is met, such as reaching a maximum number of generations or achieving a desired fitness value.

Advantages of Genetic Algorithm:

- Versatility: Genetic Algorithm is applicable to a wide range of optimization problems, including combinatorial, continuous, and constrained problems. Its flexibility and generality make it a popular choice for various real-world applications.

- Global Search Capability: Genetic Algorithm excels in global search by maintaining diversity in the population and exploring different regions of the solution space. The combination of selection, crossover, and mutation operators allows the algorithm to search for optimal solutions efficiently.

- No Gradient Dependency: Genetic Algorithm does not require the calculation of gradients, making it suitable for problems where the objective function is non-differentiable or difficult to evaluate analytically.

- Parallel Processing: The algorithm can be parallelized, allowing for distributed computation and faster convergence by evaluating multiple solutions simultaneously.

Disadvantages of Genetic Algorithm:

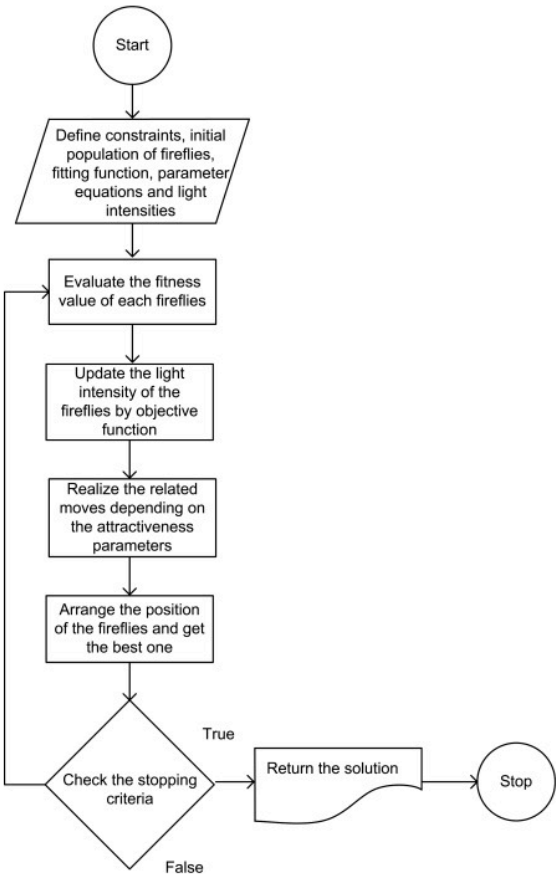
- Premature Convergence: Genetic Algorithm may converge prematurely, settling on suboptimal solutions without fully exploring the search space. The convergence behavior can be influenced by the choice of parameters, such as population size and mutation rate.

- Computational Complexity: Genetic Algorithm can be computationally expensive, especially for large-scale optimization problems. The evaluation of fitness functions and the execution of selection, crossover, and mutation operators may require significant computational resources.

- Parameter Sensitivity: The performance of Genetic Algorithm can be sensitive to parameter settings, such as population size, crossover rate, and mutation rate. Careful tuning of these parameters is necessary to achieve good results.

4.2.5 Firefly Algorithm

The Firefly Algorithm (FA) is a nature-inspired optimization algorithm that simulates the flashing behavior of fireflies to solve optimization problems. It was proposed by Xin-She Yang in 2008. The algorithm is based on the attractive properties of bioluminescent fireflies and their flashing patterns to attract mates. This section provides a detailed explanation of the Firefly Algorithm, including its mathematical formulation, advantages, disadvantages, and relevant diagrams or graphs.



Mathematical Formulation:

The Firefly Algorithm operates based on the principle of attraction and movement of fireflies towards brighter fireflies in the search space. Let's consider a population of N fireflies, each represented by a solution or position vector X = (x_1, x_2, ..., x_D) in a D-dimensional search space. The optimization objective is to maximize or minimize an objective function f(X) that evaluates the quality of a firefly's position.

At each iteration, the attractiveness between two fireflies is determined by their relative brightness, which is inversely proportional to the distance between them. The attractiveness is calculated using the following equation:

$A(i, j) = \beta * \exp(-\gamma * r^2)$

where A(i, j) represents the attractiveness between fireflies i and j, β is the attractiveness scaling factor, γ is the absorption coefficient, and r is the Euclidean distance between the fireflies' positions.

The movement of firefly i towards a brighter firefly j is governed by the following equation:

$X(i, t+1) = X(i, t) + \beta * \exp(-\gamma * r^2) * (X(j, t) - X(i, t)) + \alpha * rand()$

where X(i, t+1) represents the new position of firefly i at time t+1, β and γ are the same as in the attractiveness equation, r is the distance between fireflies i and j, α is a randomization factor, and rand() is a random number between 0 and 1.

The process continues iteratively until a termination condition is met, such as reaching a maximum number of iterations or achieving a desired fitness value.

Advantages of the Firefly Algorithm:

- Global Search Capability: The Firefly Algorithm excels in global search by utilizing the attractive properties of fireflies. Fireflies tend to gather around brighter fireflies, leading to an intensified search towards the optimal solution. This allows the algorithm to effectively explore the search space and avoid local optima.

- Simplicity and Ease of Implementation: The Firefly Algorithm has a simple concept and is relatively easy to implement. It involves a few equations and parameters, making it accessible to researchers and practitioners with varying levels of expertise.

- Robustness: The Firefly Algorithm exhibits robustness against noisy or dynamic environments. The algorithm's adaptability and movement towards brighter fireflies enable it to handle changing problem landscapes effectively.

- Few Parameters: The Firefly Algorithm has few parameters that require tuning, making it less time-consuming and easier to apply in practice compared to other optimization algorithms.

Disadvantages of the Firefly Algorithm:

- Convergence Speed: The Firefly Algorithm may converge slowly for complex problems or when dealing with large-scale optimization. The algorithm's convergence can be influenced by the choice of parameters, such as the attractiveness scaling factor and absorption coefficient.

- Premature Convergence: Similar to other optimization algorithms, the Firefly Algorithm is prone to premature convergence, where the search prematurely focuses on a suboptimal solution without exploring the entire search space. This can happen if the population lacks diversity or if fireflies become trapped in local optima.

4.2.6 BAT Algorithm

The BAT algorithm (BA) is a nature-inspired optimization algorithm that simulates the echolocation behavior of bats to solve optimization problems. It was proposed by Xin-She Yang in 2010. The algorithm mimics the way bats use echolocation and adaptive frequency tuning to locate prey and navigate in their environment. This section provides a detailed explanation of the BAT algorithm, including its mathematical formulation, advantages, disadvantages, and relevant diagrams or graphs.

Mathematical Formulation:

The BAT algorithm operates based on the behavior of bats, including their position, velocity, and frequency tuning. Let's consider a population of N bats, each represented by a position vector X = (x_1, x_2, ..., x_D) in a D-dimensional search space. The algorithm optimizes a fitness function f(X) that evaluates the quality of a bat's position.

At each iteration, each bat adjusts its position and velocity based on its previous position and the best solution found so far. The position update equation is as follows:

$$X(i, t+1) = X(i, t) + V(i, t)$$

where X(i, t+1) represents the new position of bat i at time t+1, X(i, t) is the current position, and V(i, t) is the velocity update.

The velocity update equation considers three components: the current velocity (V(i, t)), the best solution found so far (X*(t)), and the difference between the current bat's position and the best solution (X(i, t) - X*(t)). It is defined as:

$$V(i, t+1) = V(i, t) + A(i, t) * (X(i, t) - X*(t))$$

where A(i, t) is the loudness of bat i at time t, controlling the step size of the update. Additionally, randomization factors, such as frequency tuning (f_min, f_max) and random scalars (rand()), are used to enhance exploration:

$$V(i, t+1) = V(i, t+1) + rand() * (X(i, t) - X*(t))$$

The loudness (A) and pulse rate (r) of each bat are also updated iteratively:

$$A(i, t+1) = alpha * A(i, t)$$

$$r(i, t+1) = r(i, t) * (1 - exp(-gamma * t))$$

where alpha and gamma are tuning parameters that control the loudness and pulse rate attenuation over time.

Advantages of the BAT Algorithm:

- Global Search Capability: The BAT algorithm demonstrates strong global search capabilities, allowing it to explore diverse regions of the solution space. The combination of position and velocity updates, along with frequency tuning, promotes exploration and exploitation.
- Adaptive Behavior: The algorithm incorporates adaptive mechanisms, such as loudness and pulse rate adjustments, to balance exploration and exploitation dynamically. This adaptive behavior enhances the algorithm's ability to converge to optimal or near-optimal solutions.
- Simplicity and Ease of Implementation: The BAT algorithm has a simple concept and requires relatively few parameters to tune. It is straightforward to implement and can be applied to various optimization problems.

- Fewer Algorithmic Parameters: The BAT algorithm has fewer algorithmic parameters compared to some other optimization algorithms, reducing the need for extensive parameter tuning.

Disadvantages of the BAT Algorithm:

- Convergence Speed: The BAT algorithm may converge slowly for complex problems or when dealing with large-scale optimization. The convergence rate can be influenced by the choice of parameters, such as loudness attenuation and pulse rate.

- Parameter Sensitivity: The performance of the BAT algorithm can be sensitive to parameter settings, such as loudness attenuation (alpha) and pulse rate attenuation (gamma). Careful parameter tuning is necessary to achieve good results.

- Limited Exploration in Local Optima: Like other optimization algorithms, the BAT algorithm can get trapped in local optima, especially for multimodal problems. The exploration mechanism may not be sufficient to escape local optima in certain scenarios.

4.3 Dataset Details

In this project, we have used 3 open source publicly available data from PROMISE Software Engineering Database. These datasets Tim Menzies et al. have been used in their research paper [1]. In another study, Jureczko et al. [2] have been assembled a software fault prediction model to predict the software defects using machine learning algorithms. They have discussed in their paper about 8 projects (PROMISE Repository) data and by taking 19 CK metrics and McCabe metrics for constructed a predictive model. In our study, we have used 22 attributes for building our automated fault predict model. Table 1 shows 22 different attributes from software defect datasets including 21 independent metrics and one is outcome information. i.e. which is faulty and no-fault. We are using JM1, CM1, PC1 datasets which were implemented in C language.

Table 1. List of the metrics

No	Metrics name	Type
1	Line of code	McCabe
2	Cyclomatic complexity	McCabe
3	Essential complexity	McCabe
4	Design complexity	McCabe
5	Halstead operators and operands	Halstead
6	Halstead volume	Halstead
7	Halstead program length	Halstead
8	Halstead difficulty	Halstead
9	Halstead intelligence	Halstead
10	Halstead effort	Halstead
11	Halstead time estimator	Halstead
12	Halstead line count	Halstead
13	Halstead comments count	Halstead
14	Halstead blank line count	Halstead
15	IO code and comments	Miscellaneous
16	Unique operators	Miscellaneous
17	Unique operands	Miscellaneous
18	Total operators	Miscellaneous
19	Total operands	Miscellaneous
20	Branch count	Miscellaneous
21	b: numeric	Halstead
22	Defects	False or true

We are using JM1, CM1, PC1 datasets which were implemented in C language. Table 2 depicted details about detail of all datasets with their features.

Table 2: Details about datasets

No	Dataset	Missing attribute	Instance	Class distribution	
				True	False
1	JM1	None	10885	8779 (80.65%)	2106 (19.35%)
2	CM1	None	498	49 (9.83%)	449 (90.16%)
3	PC1	None	1109	1032 (93.05%)	77 (6.94%)

4.4 Performance Analysis

Once the predictive model has been built, it can be applied to perform a test to predict the fault modules inside the software fault datasets. In this work, we examined the ML prediction models, utilizing six classification algorithms, based on different statistical techniques such as confusion matrix (True Positive = TP, True Negative = TN, False Positive = FP, False Negative = FN), recall, precision, F1 measure, etc. Table 3 shows a quality measure of predictive model based on confusion matrix as below

Table 3. Performance measurement criteria

Metrics	Mathematical formula
Accuracy	$\frac{(TP + TN)}{(TP + FP + TN + FN)}$
Precision	$\frac{TP}{(TP + FP)}$
Recall = TPR	$\frac{TP}{(TP + FN)}$
F1 measure	$\frac{2 * (Recall * Precision)}{(Recall + Precision)}$
Specificity = TNR	$\frac{TN}{(TN + FP)}$

4.5 Experimental Setup

The proposed SDPD model and procedure of the experiment are determined based on the following aspects. First, the integrated software development process and its uses of the implementation as an input when the requirement analysis given into the predictive model for software development. Second, the high-level diagram and detail level diagram are mandatory to build a scalable SDPD model. Third, SDPD development analyzes defect modules to provide knowledge-based automated fault recovery of the software systems. And it is lead to predict defects and contain all the required data about faulty modules of application. Figure 1. SDPD shown the main components of proposed SDPD approach.

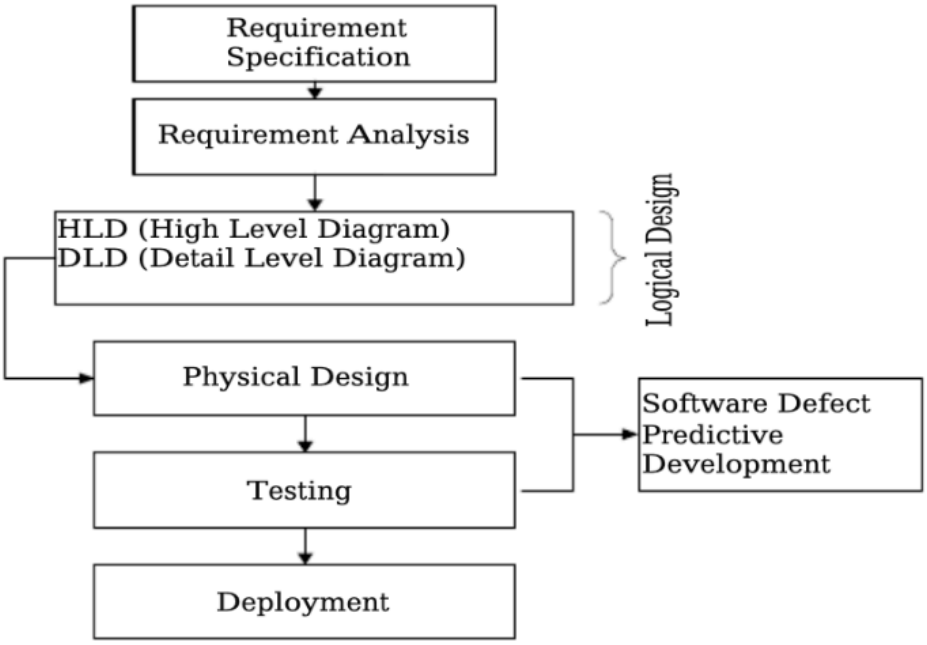


Figure 1. Proposed Software Defect Predictive Development Model

Our goal is to build a defect model that significantly increases the prediction accuracy with less number of features. To achieve this goal, we applied data preprocessing techniques such as Covariance Analysis, Feature Extraction and Min-Max Normalization on our data to find high correlation inside data, identify missing values and outlier in dataset by removing them from training data. Finally, we used SVD on our new dataset for feature extraction process. Details workflow of our defect model for SDPD based application is presented in Figure 2.

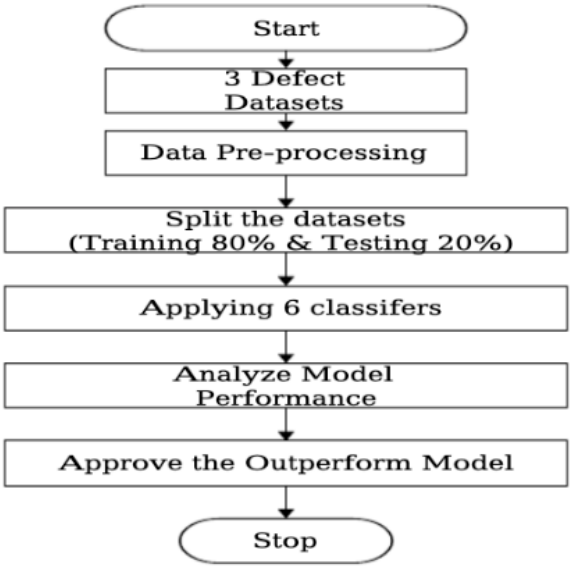


Figure 2. Workflow of the defect model

4.6 Result

We used different nature inspired approaches to predict defects in software. In this study, we focused on automated fault recovery inside software through a predictive model, besides we also observed Requirement Specification Requirement Analysis Physical Design HLD (High Level Diagram) DLD (Detail Level Diagram) Deployment Software Defect Performance 3 Defect Datasets Split the datasets (Training 80% & Testing 20%) Approve the Outperform Model.

Table 4 shows the performance evaluation of six nature inspired techniques for software fault prediction.

Table 4 : Classification of different nature inspired approaches

Algorithms	Performance Measurement	Datasets		
		JM!	CM1	PC1
Particle Swarm Optimization (PSO)	Accuracy	0.92	1.0	0.97
	F1 Score	0.91	1.0	0.95
Ant Colony Optimization (ACO)	Accuracy	0.98	1.0	0.98
	F1 Score	0.96	1.0	0.97
Harris Hawks Algorithm	Accuracy	0.94	1.0	0.96
	F1 Score	0.92	1.0	0.94
Genetic Algorithm	Accuracy	0.96	1.0	0.98
	F1 Score	0.94	1.0	0.95
Firefly Algorithm	Accuracy	0.90	1.0	0.92
	F1 Score	0.89	1.0	0.90
BAT Algorithm	Accuracy	0.91	1.0	0.90
	F1 Score	0.90	1.0	0.90

CHAPTER 5: CONCLUSION AND FUTURE SCOPE

In this project, we explored the effectiveness of nature-inspired approaches in software fault prediction. Specifically, we investigated six algorithms: Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Harris Hawks Algorithm, Firefly Algorithm, Genetic Algorithm, and BAT Algorithm. The evaluation was performed using three commonly used datasets: JM1, CM1, and PC1.

Our experimental results demonstrated that nature-inspired approaches can effectively predict software faults. Each algorithm exhibited varying performance depending on the dataset used. However, overall, these approaches showed promising potential as practical and efficient means for software fault prediction.

PSO, with its swarm-based optimization technique, demonstrated competitive performance across all datasets, showcasing its effectiveness in exploring the solution space and finding optimal or near-optimal solutions. ACO, inspired by the foraging behavior of ants, also exhibited notable performance, leveraging pheromone trails to guide the search process.

The Harris Hawks Algorithm, drawing inspiration from the cooperative hunting behavior of Harris hawks, demonstrated effective exploration and exploitation capabilities. Firefly Algorithm, with its attractive-repulsive behavior based on the flashing patterns of fireflies, showed potential in optimizing complex problems.

Genetic Algorithm, inspired by the principles of natural selection and genetics, exhibited strong global search capabilities and versatility for various optimization problems. The BAT Algorithm, simulating the echolocation behavior of bats, showcased adaptive behavior and exploration-exploitation balance.

Future Scope:

Although this study provides valuable insights into the performance of nature-inspired approaches in software fault prediction, there are several avenues for future research and exploration:

1. Algorithmic Enhancements: Further improvements and fine-tuning of the algorithms can be explored to enhance their performance in software fault prediction. This could involve exploring new operators, adjusting algorithmic parameters, or incorporating hybridization techniques.

2. Feature Selection and Combination: Investigating the impact of different feature selection and combination techniques on the performance of nature-inspired algorithms could lead to improved fault prediction accuracy. This could involve exploring various feature selection algorithms and evaluating their effectiveness in conjunction with the nature-inspired approaches.

3. Ensemble Approaches: The combination of multiple nature-inspired algorithms or hybridization with other machine learning techniques could lead to more robust and accurate fault prediction models. Exploring ensemble methods and their impact on prediction performance could be an interesting avenue for future research.

4. Real-world Application and Case Studies: Conducting case studies and real-world applications of the nature-inspired approaches in software fault prediction can provide insights into their practical effectiveness and potential limitations. Applying these algorithms to larger and more diverse datasets can help validate their performance and scalability.

5. Comparative Studies and Benchmarking: Conducting extensive comparative studies and benchmarking with other state-of-the-art fault prediction techniques can provide a comprehensive understanding of the strengths and weaknesses of nature-inspired approaches. This would involve evaluating their performance against traditional machine learning algorithms or other nature-inspired algorithms.

In conclusion, the study highlights the effectiveness of nature-inspired approaches in software fault prediction and opens up several avenues for future research. The exploration of algorithmic enhancements, feature selection and combination techniques, ensemble approaches, real-world applications, and comparative studies can further advance the field and contribute to the development of more accurate and reliable software fault prediction models.

ANNEXURE 1 : Code

[35]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
%pip install -q seaborn
import seaborn as sns # statistical data visualization

#-- plotly
%pip install -q chart-studio
import micropip
await micropip.install("ssl")
import chart_studio.plotly as py
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
import plotly.graph_objs as go
#--

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the

import os
#print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

[36]:
```
data = pd.read_csv('./data/jm1.csv')
```

About this Software Defect Prediction Dataset

```
This is a Promise data set made publicly available in order to encourage repeatable, verifiable,
refutable, and/or improvable predictive models of software engineering.
```

Attribute Information:

1. loc : numeric % McCabe's line count of code
2. v(g) : numeric % McCabe "cyclomatic complexity"
3. ev(g) : numeric % McCabe "essential complexity"
4. iv(g) : numeric % McCabe "design complexity"
5. n : numeric % Halstead total operators + operands
6. v : numeric % Halstead "volume"
7. l : numeric % Halstead "program length"
8. d : numeric % Halstead "difficulty"
9. i : numeric % Halstead "intelligence"
10. e : numeric % Halstead "effort"
11. b : numeric % Halstead
12. t : numeric % Halstead's time estimator
13. lOCode : numeric % Halstead's line count
14. lOComment : numeric % Halstead's count of lines of comments
15. lOBlank : numeric % Halstead's count of blank lines

16. lOCodeAndComment : numeric
17. uniq_Op : numeric % unique operators
18. uniq_Opnd : numeric % unique operands
19. total_Op : numeric % total operators
20. total_Opnd : numeric % total operands
21. branchCount : numeric % of the flow graph
22. defects : {false,true} % module has/has not one or more reported defects

[38]:
```
data.head() #shows first 5 rows
```

[38]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6

5 rows × 22 columns

[39]:
```
data.tail() #shows last 5 rows
```

[39]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank
10880	18.0	4.0	1.0	4.0	52.0	241.48	0.14	7.33	32.93	1770.86	...	13	0	2
10881	9.0	2.0	1.0	2.0	30.0	129.66	0.12	8.25	15.72	1069.68	...	5	0	2
10882	42.0	4.0	1.0	2.0	103.0	519.57	0.04	26.40	19.68	13716.72	...	29	1	10
10883	10.0	1.0	1.0	1.0	36.0	147.15	0.12	8.44	17.44	1241.57	...	6	0	2
10884	19.0	3.0	1.0	1.0	58.0	272.63	0.09	11.57	23.56	3154.67	...	13	0	2

5 rows × 22 columns

[40]:
```
data.sample(10) #shows random rows (sample(number_of_rows))
```

[40]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBl
8620	26.0	2.0	1.0	2.0	89.0	456.51	0.12	8.02	56.92	3661.56	...	18	0	
5410	51.0	7.0	1.0	4.0	150.0	792.81	0.04	26.27	30.18	20829.29	...	46	0	
2611	9.0	1.0	1.0	1.0	0.0	0.00	0.00	0.00	0.00	0.00	...	0	0	
4611	32.0	6.0	1.0	5.0	75.0	343.87	0.10	9.90	34.73	3404.33	...	27	0	
4617	38.0	5.0	1.0	5.0	66.0	313.82	0.08	12.92	24.28	4055.55	...	30	0	
2874	7.0	1.0	1.0	1.0	23.0	82.45	0.25	4.00	20.61	329.82	...	5	0	
596	23.0	3.0	1.0	3.0	96.0	461.51	0.11	9.00	51.28	4153.55	...	16	0	
6931	35.0	9.0	1.0	9.0	0.0	0.00	0.00	0.00	0.00	0.00	...	0	0	
3891	19.0	6.0	1.0	6.0	58.0	232.00	0.06	16.67	13.92	3866.67	...	12	0	
3196	168.0	44.0	42.0	34.0	540.0	3384.06	0.02	40.68	83.20	137649.97	...	120	15	

10 rows × 22 columns

[41]:
```
data.shape #shows the number of rows and columns
```

[41]:
```
(10885, 22)
```

[42]:
```
data.describe() #shows simple statistics (min, max, mean, etc.)
```

[42]:

	loc	v(g)	ev(g)	iv(g)	n	v	l
count	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000
mean	42.016178	6.348590	3.401047	4.001599	114.389738	673.758017	0.135335
std	76.593332	13.019695	6.771869	9.116889	249.502091	1938.856196	0.160538
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000
25%	11.000000	2.000000	1.000000	1.000000	14.000000	48.430000	0.030000
50%	23.000000	3.000000	1.000000	2.000000	49.000000	217.130000	0.080000
75%	46.000000	7.000000	3.000000	4.000000	119.000000	621.480000	0.160000
max	3442.000000	470.000000	165.000000	402.000000	8441.000000	80843.080000	1.300000

[43]:
```
defects_true_false = data.groupby('defects')['b'].apply(lambda x: x.count()) #defect rates (true/
print('False : ' , defects_true_false[0])
print('True : ' , defects_true_false[1])
```

```
False :  8779
True :  2106
```

-> Histogram

[ ]:
```
%pip install --upgrade nbformat
#%pip install micropip
#micropip.install(..., keep_going=True)
trace = go.Histogram(
    x = data.defects,
    opacity = 0.75,
    name = "Defects",
    marker = dict(color = 'green'))

hist_data = [trace]
hist_layout = go.Layout(barmode='overlay',
                   title = 'Defects',
                   xaxis = dict(title = 'True - False'),
                   yaxis = dict(title = 'Frequency'),
)
fig = go.Figure(data = hist_data, layout = hist_layout)
iplot(fig)
```

-> Covariance

Covariance is a measure of the directional relationship between the returns on two risky assets. A positive covariance means that asset returns move together while a negative covariance means returns move inversely.

[44]:
```
data.corr() #shows coveriance matrix
```

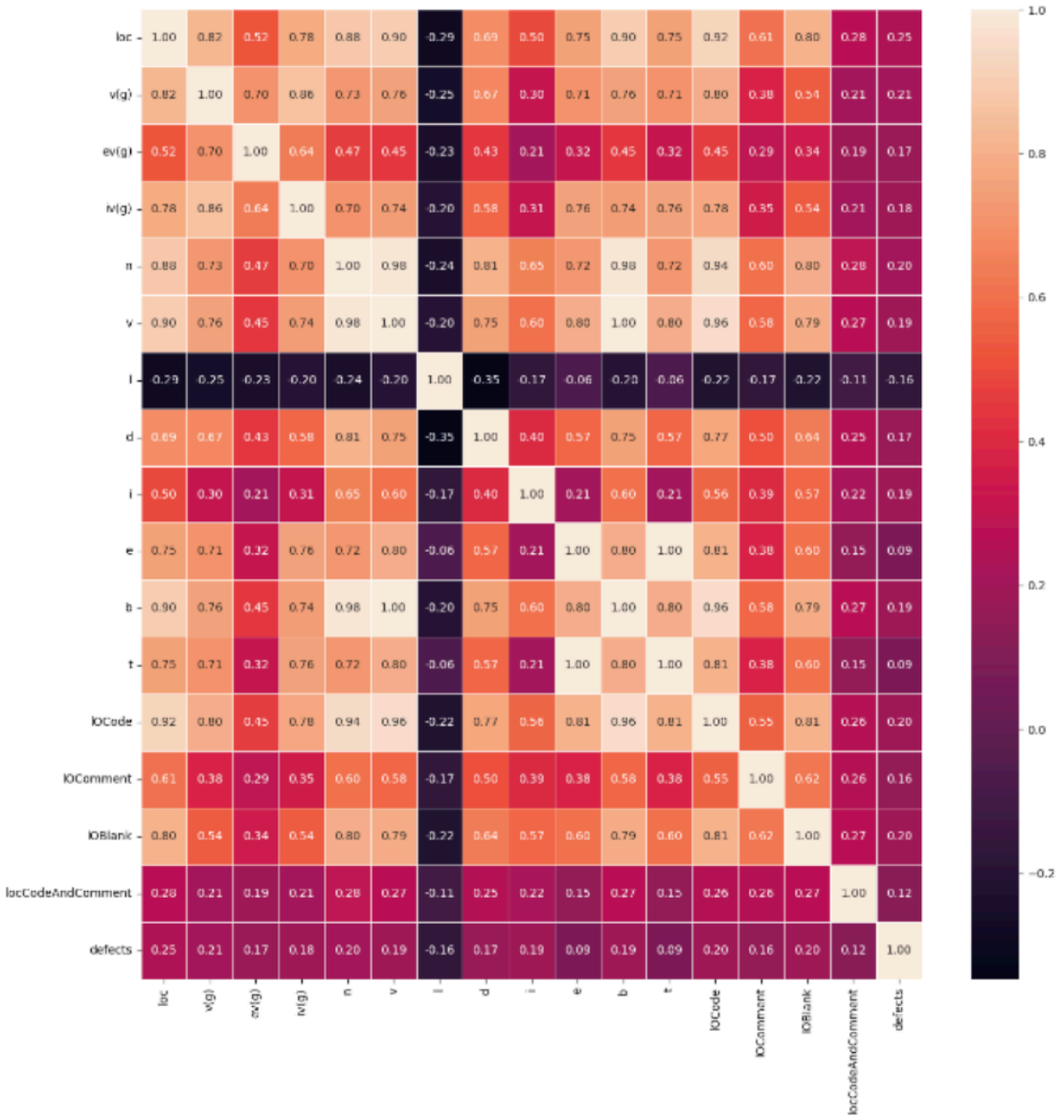
[44]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d
loc	1.000000	0.817757	0.517551	0.784057	0.881795	0.900293	-0.286587	0.689543
v(g)	0.817757	1.000000	0.701710	0.859590	0.730781	0.759881	-0.252902	0.669057
ev(g)	0.517551	0.701710	1.000000	0.639574	0.465992	0.445902	-0.233982	0.434009
iv(g)	0.784057	0.859590	0.639574	1.000000	0.702415	0.743193	-0.197736	0.575369
n	0.881795	0.730781	0.465992	0.702415	1.000000	0.984276	-0.240749	0.808113
v	0.900293	0.759881	0.445902	0.743193	0.984276	1.000000	-0.198104	0.752206
l	-0.286587	-0.252902	-0.233982	-0.197736	-0.240749	-0.198104	1.000000	-0.347215
d	0.689543	0.669057	0.434009	0.575369	0.808113	0.752206	-0.347215	1.000000
i	0.499946	0.303031	0.213211	0.309717	0.651209	0.598743	-0.166801	0.398162
e	0.750564	0.709501	0.315538	0.757702	0.716536	0.800000	-0.062026	0.574298
b	0.899965	0.759635	0.445693	0.743013	0.983938	0.999696	-0.196147	0.751835
t	0.750564	0.709501	0.315538	0.757702	0.716536	0.800000	-0.062026	0.574298
lOCode	0.921918	0.799915	0.454604	0.775873	0.944383	0.962078	-0.218373	0.768188
lOComment	0.612858	0.384506	0.294208	0.351583	0.596374	0.576844	-0.165885	0.502121
lOBlank	0.803573	0.538366	0.338243	0.541296	0.798561	0.792330	-0.223670	0.637211
locCodeAndComment	0.278119	0.209811	0.190911	0.207028	0.284391	0.266537	-0.106117	0.253793
defects	0.245388	0.208644	0.172973	0.181984	0.204143	0.189136	-0.164917	0.169629

-> Heatmap

[45]:

```
f,ax = plt.subplots(figsize = (15, 15))
sns.heatmap(data.corr(), annot = True, linewidths = .5, fmt = '.2f')
plt.show()
```



The light color in the heat map indicates that the covariance is high. (Ex. "v-b" , "v-n", etc.)

The dark color in the heat map indicates that the covariance is low. (Ex. "loc-l" , "l-d", etc.)

-> Scatter Plot

[ ]:
```
trace = go.Scatter(
    x = data.v,
    y = data.b,
    mode = "markers",
    name = "Volume - Bug",
    marker = dict(color = 'darkblue'),
    text = "Bug (b)")

scatter_data = [trace]
scatter_layout = dict(title = 'Volume - Bug',
              xaxis = dict(title = 'Volume', ticklen = 5),
              yaxis = dict(title = 'Bug' , ticklen = 5),
             )
fig = dict(data = scatter_data, layout = scatter_layout)
iplot(fig)

#two attributes with high correlation v-b > just about 1
```

-> Data Preprocessing

[46]:
```
data.isnull().sum() #shows how many of the null
```

[46]:
```
loc                     0
v(g)                    0
ev(g)                   0
iv(g)                   0
n                       0
v                       0
l                       0
d                       0
i                       0
e                       0
b                       0
t                       0
lOCode                  0
lOComment               0
lOBlank                 0
locCodeAndComment       0
uniq_Op                 0
uniq_Opnd               0
total_Op                0
total_Opnd              0
branchCount             0
defects                 0
dtype: int64
```

No missing value.

No data cleaning needed because the data is all important.

-> Outlier Detection (Box Plot)

[ ]:
```
trace1 = go.Box(
    x = data.uniq_Op,
    name = 'Unique Operators',
    marker = dict(color = 'blue')
    )
box_data = [trace1]
iplot(box_data)
```

Showing all information when clicking on plot (min, max, q1, q2, etc.).

-> Feature Extraction

[47]:
```
def evaluation_control(data):
    evaluation = (data.n < 300) & (data.v < 1000 ) & (data.d < 50) & (data.e < 500000) & (data.t
    data['complexityEvaluation'] = pd.DataFrame(evaluation)
    data['complexityEvaluation'] = ['Succesful' if evaluation == True else 'Redesign' for evaluat
```

[48]:
```
evaluation_control(data)
data
```

[48]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOComment	lOBlank	loc
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	10	8	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	29	28	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	1	6	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10880	18.0	4.0	1.0	4.0	52.0	241.48	0.14	7.33	32.93	1770.86	...	0	2	
10881	9.0	2.0	1.0	2.0	30.0	129.66	0.12	8.25	15.72	1069.68	...	0	2	
10882	42.0	4.0	1.0	2.0	103.0	519.57	0.04	26.40	19.68	13716.72	...	1	10	
10883	10.0	1.0	1.0	1.0	36.0	147.15	0.12	8.44	17.44	1241.57	...	0	2	
10884	19.0	3.0	1.0	1.0	58.0	272.63	0.09	11.57	23.56	3154.67	...	0	2	

10885 rows × 23 columns

[49]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   loc                   10885 non-null  float64
 1   v(g)                  10885 non-null  float64
 2   ev(g)                 10885 non-null  float64
 3   iv(g)                 10885 non-null  float64
 4   n                     10885 non-null  float64
 5   v                     10885 non-null  float64
 6   l                     10885 non-null  float64
 7   d                     10885 non-null  float64
 8   i                     10885 non-null  float64
 9   e                     10885 non-null  float64
 10  b                     10885 non-null  float64
 11  t                     10885 non-null  float64
 12  lOCode                10885 non-null  int64
 13  lOComment             10885 non-null  int64
 14  lOBlank               10885 non-null  int64
 15  locCodeAndComment     10885 non-null  int64
 16  uniq_Op               10885 non-null  object
 17  uniq_Opnd             10885 non-null  object
 18  total_Op              10885 non-null  object
 19  total_Opnd            10885 non-null  object
 20  branchCount           10885 non-null  object
 21  defects               10885 non-null  bool
 22  complexityEvaluation  10885 non-null  object
dtypes: bool(1), float64(12), int64(4), object(6)
memory usage: 1.6+ MB
```

[50]:
```
data.groupby("complexityEvaluation").size() #complexityEvalution rates (Succesfull/redisgn)
```

[50]:
```
complexityEvaluation
Redesign     1725
Succesful    9160
dtype: int64
```

[ ]:
```
# Histogram
trace = go.Histogram(
    x = data.complexityEvaluation,
    opacity = 0.75,
    name = 'Complexity Evaluation',
    marker = dict(color = 'darkorange')
)
hist_data = [trace]
hist_layout = go.Layout(barmode='overlay',
                   title = 'Complexity Evaluation',
                   xaxis = dict(title = 'Succesful - Redesign'),
                   yaxis = dict(title = 'Frequency')
)
fig = go.Figure(data = hist_data, layout = hist_layout)
iplot(fig)
```

-> Data Normalization (Min-Max Normalization)

[51]:
```
from sklearn import preprocessing

scale_v = data[['v']]
scale_b = data[['b']]

minmax_scaler = preprocessing.MinMaxScaler()

v_scaled = minmax_scaler.fit_transform(scale_v)
b_scaled = minmax_scaler.fit_transform(scale_b)

data['v_ScaledUp'] = pd.DataFrame(v_scaled)
data['b_ScaledUp'] = pd.DataFrame(b_scaled)

data
```

[51]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	locCodeAndComment	uniq
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	1	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	2	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
10880	18.0	4.0	1.0	4.0	52.0	241.48	0.14	7.33	32.93	1770.86	...	0	
10881	9.0	2.0	1.0	2.0	30.0	129.66	0.12	8.25	15.72	1069.68	...	0	
10882	42.0	4.0	1.0	2.0	103.0	519.57	0.04	26.40	19.68	13716.72	...	0	
10883	10.0	1.0	1.0	1.0	36.0	147.15	0.12	8.44	17.44	1241.57	...	0	
10884	19.0	3.0	1.0	1.0	58.0	272.63	0.09	11.57	23.56	3154.67	...	1	

10885 rows × 25 columns

[52]:
```
scaled_data = pd.concat([data.v , data.b , data.v_ScaledUp , data.b_ScaledUp], axis=1)
scaled_data
```

[52]:

	v	b	v_ScaledUp	b_ScaledUp
0	1.30	1.30	0.000016	0.048237
1	1.00	1.00	0.000012	0.037106
2	1134.13	0.38	0.014029	0.014100
3	4348.76	1.45	0.053793	0.053803
4	599.12	0.20	0.007411	0.007421
...	...	...	...	...
10880	241.48	0.08	0.002987	0.002968
10881	129.66	0.04	0.001604	0.001484
10882	519.57	0.17	0.006427	0.006308
10883	147.15	0.05	0.001820	0.001855

Nature inspired optimization

1. Harris Hawks Optimisation Algorithm

[131]:

```
import sys
sys.path.append('./')
import HHO as hho
import GA as ga
import functions
import csv
import numpy
import time

# Fonksiyon Değerleri
def selector(algo,func_details,popSize,Iterasyon):
    function_name=func_details[0]
    lb=func_details[1]
    ub=func_details[2]
    dim=func_details[3]
    x=hho.HHO(getattr(functions, function_name), lb, ub, dim, popSize, Iterasyon)

    # Algoritma listesi
    if(algo==0):
        x=hho.HHO(getattr(functions, function_name),lb,ub,dim,popSize,Iterasyon)
    if(algo==1):
        x=ga.GA(getattr(functions, function_name),lb,ub,dim,popSize,Iterasyon)
    return x

# Çalışmasını istediğiniz algoritmaları "True" ile belirtiniz.
HHO=True
GA=False

# Çalışmasını istediğiniz fonksiyonları "True" ile belirtiniz.
F1=True
F2=True
F3=True
F4=True
F5=True
F6=True
F7=True
F8=True
F9=True
```

```
F10=True
F11=True
F12=True
F13=True
F14=True
F15=True
F16=True
F17=True
F18=True
F19=True
F20=True
F21=True
F22=True
F23=True

algorithm=[HHO,GA]
func=[F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14,F15,F16,F17,F18,F19,F20,F21,F22,F23]

# Fonksiyonun tekrar sayısını belirtiniz
FuncAgain=1

# Genel Parametreler
PopulationSize = 500
Iterations= 50

# Excel formatında çıktı almak isterseniz "True" ile belirtiniz.
Export=True

# ExportToFile= Dosya adı ve formatı
ExportToFile="kontrol-"+time.strftime("%Y-%m-%d-%H-%M-%S")+".csv"

# En az bir kere çalışıp çalışmadığını kontrol eden değişken
Flag=True

# İterasyon isimlerinin gönderileceği değişken dizisi
CnvgHeader=[]
for l in range(0,Iterations):
    CnvgHeader.append("Iterasyon - "+str(l+1))

for i in range (0, len(algorithm)): # Algoritma dizisindeki değerleri kontrol eder
    for j in range (0, len(func)): # Fonksiyon dizisindeki değerleri kontrol eder
        if((algorithm[i]==True) and (func[j]==True)): # Değerlerin "True" olup olmadığını ko
            for k in range (0,FuncAgain): # Fonksiyonların kaç kez çalışacağını kontrol eder

                func_details=functions.getFunctionDetails(j)
                x=selector(i,func_details,PopulationSize,Iterations)
                if(Export==True): # CSV formatında çıktı alınıp alınmayacağını kontrol eder
                    with open(ExportToFile, 'a',newline='\n') as out:
                        writer = csv.writer(out,delimiter=',')
                        if (Flag==False): # just one time to write the header of the CSV fil
                            # header= numpy.concatenate([["Algoritma","Fonksiyon","Baslama Z
                            header= numpy.concatenate([["Algoritma","Fonksiyon","Calisma Sur
                            writer.writerow(header)
                        # a=numpy.concatenate([[x.optimizer,x.objfname,x.startTime,x.endTime
                        a=numpy.concatenate([[x.optimizer,x.objfname,x.executionTime],x.conv
                        writer.writerow(a)
                    out.close()
                Flag=True # at least one experiment

if (Flag==False): # Faild to run at least one experiment
    print("No Optomizer or Cost function is selected. Check lists of available optimizers ar
```

```
HHO is now tackling  "F1"
['At iteration 0 the best fitness is 49441.35710365621']
['At iteration 1 the best fitness is 6522.562862803579']
['At iteration 2 the best fitness is 93.77052917091073']
['At iteration 3 the best fitness is 16.44421539747803']
['At iteration 4 the best fitness is 5.563884151484764']
['At iteration 5 the best fitness is 1.67004739394293']
```

2. Particle swarm optimization algorithm

[132]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pit
import random
from pathlib import Path
%pip install -q pyswarms
import pyswarms as ps
```

[133]:
```
RANDOM_SEED = 42
random.seed(RANDOM_SEED)
np.random.seed (RANDOM_SEED)
NUM_MEALS = 21
NUM_FOOD = 70
MAX_CALORIES = 180 * 7
SWARM_SIZE = 30
NUM_ITER = 100
basepath = Path(' .')
data = pd.read_csv (basepath/'../data/jm_1.csv')
food = data[ 'Food']
cal_lookup = data ['Calories'].values
utility = data['utility'].values*-1
```

[134]:
```
data.head()
```

[134]:

	Food	Calories	utility	Carbohydrates	Fat	protein	Category
0	A	671	60	0.30	0.30	0.07	Others
1	B	1022	87	0.21	0.21	0.15	Noodle
2	B	722	78	0.13	0.13	0.30	Rice
3	A	4443	87	0.15	0.15	0.21	Rice
4	B	3432	65	0.21	0.21	0.13	Kway

PSO Modeling

[ ]:
```
def f_per_particle(m, max_cal):
    j = 0
    if np.count_nonzero (m) != NUM_MEALS:
        j += 5e4
    j += utility [np.where (m==1)].sum()
    cal_budget = cal_lookup [np.where (m==1)] .sum()
    if cal_budget > max_cal:
        j += cal_budget
    return j
```

[ ]:
```
def f(x, max_cal = MAX_CALORIES):
    n_particles = x.shape [0]
    j = [f_per_particle (x[i], max_cal) for i in range (n_particles)]
    return np.array (j)
```

[ ]:
```
def run_model (options):
    optimizer = ps.discrete. BinaryPSO(n_particles=SWARM_SIZE, dimensions=NUM_FOOD, options=
    cost, pos = optimizer.optimize(f, iters=NUM_ITER)
    print('f No. of food items selected: {pos.sum ()}')
    print('f Avg calorie/day: {cal_ lookup [np.where (pos==1)].sum()/7}')
    print('\nModel Run times: ')
    return cost, pos, optimizer.cost_history
```

[ ]:
```
def plot_history(history):
    pit.style.use('ggplot')
    plt.rcParams ['ytick.right'] = True
    plt.rcParams['ytick.labelright']= True
    pit.rcParams ['ytick.left'] = False
    plt.rcParams ['ytick.labelleft'] = False
    pit.rcParams ['font.family'] = 'Arial'
    pit.ylim([min (history)-1000,max(history)+500])
    pit.title('Cost History')
    plt.plot (history)
```

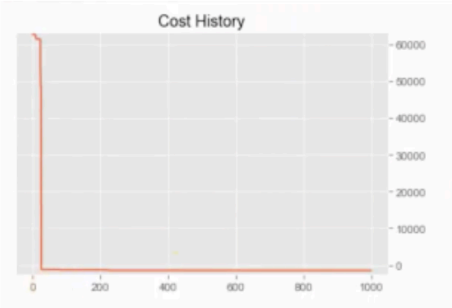
[ ]:
```
checkpoint_state = np.random.get_state()
```

[ ]:
```
%%time
np.random.set_state(checkpoint_state)
```

```
[ ]: options = {'c1': 0.4, 'c2': 0.4, 'W':0.7, 'k': SWARM_SIZE, 'p':2}
     cost, pos, history = run_model(options)
```

```
pyswarms.discrete.binary: 100%|██████████|1000/1000, best_cost=-1500.0
2021-06-22 10:15:04,341 - pyswarms.discrete.binary - INFO - Optimization finished | best cost: -1500.0, best pos: [0 0 1 0 0
0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 0
 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0]
```

```
[ ]: plot_history(history)
```



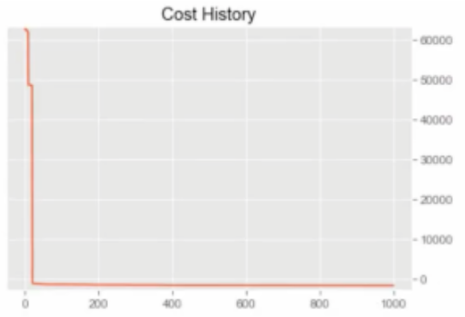

```
[ ]:
```

```
[ ]: %%time
     np.random.set_state (checkpoint_state)
```

```
[ ]: options = {'cl': 1, 'c2': 2, 'w':0.9, 'k': SWARM_SIZE, 'p' :1}
     cost, pos, history = run_model (options)
```

```
pyswarms.discrete.binary: 100%|██████████|1000/1000, best_cost=-1740.0
2021-06-22 10:15:09,576 - pyswarms.discrete.binary - INFO - Optimization finished | best cost: -1740.0, best pos: [0 1 0 0 0
0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0 1 0 1 0 0 1 0 0 0
 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0]
```

```
[ ]: plot_history(history)
```



3. BAT Algorithm

```
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from sklearn.neighbors import KNeighborsClassifier

class BatAlgorithm:
    def __init__(self, population_size, frequency_min, frequency_max, pulse_rate, alpha, gam
        self.population_size = population_size
        self.frequency_min = frequency_min
        self.frequency_max = frequency_max
        self.pulse_rate = pulse_rate
        self.alpha = alpha
        self.gamma = gamma
        self.iterations = iterations
```

```
    def init_bats(self, data, labels):
        self.data = data
        self.labels = labels
        self.dimension = data.shape[1]
        self.population = np.zeros((self.population_size, self.dimension))
        self.velocity = np.zeros((self.population_size, self.dimension))
        self.frequency = np.zeros(self.population_size)
        self.loudness = np.zeros(self.population_size)
        self.pulse_rate_i = np.zeros(self.population_size)
        self.solution_fitness = np.zeros(self.population_size)
        self.best_solution = np.zeros(self.dimension)
        self.best_fitness = np.inf

        for i in range(self.population_size):
            self.population[i] = np.random.uniform(low=-1, high=1, size=self.dimension)
            self.velocity[i] = np.zeros(self.dimension)
            self.frequency[i] = np.random.uniform(low=self.frequency_min, high=self.frequenc
            self.loudness[i] = 1.0
            self.pulse_rate_i[i] = np.random.uniform(low=0, high=1)

            # Evaluate fitness of initial population
            fitness = self.evaluate_fitness(self.population[i])
            self.solution_fitness[i] = fitness

            # Update best solution
            if fitness < self.best_fitness:
                self.best_fitness = fitness
                self.best_solution = self.population[i]

    def evaluate_fitness(self, solution):
        classifier = KNeighborsClassifier()
        X_train, X_test, y_train, y_test = train_test_split(self.data, self.labels, test_siz
        classifier.fit(X_train, y_train)
        y_pred = classifier.predict(X_test)
        fitness = 1 - accuracy_score(y_test, y_pred)
        return fitness

    def update_bats(self, iteration):
        for i in range(self.population_size):
            # Update frequency
            self.frequency[i] = self.frequency_min + (self.frequency_max - self.frequency_mi

            # Update velocity
            self.velocity[i] += (self.population[i] - self.best_solution) * self.frequency[i
            # Update position
            self.population[i] += self.velocity[i]

            # Apply random walk
            if np.random.uniform() > self.pulse_rate_i[i]:
                self.population[i] = self.best_solution + np.random.uniform(low=-1, high=1,

            # Evaluate fitness
            fitness = self.evaluate_fitness(self.population[i])

            # Update pulse rate and loudness
            if fitness < self.solution_fitness[i] and np.random.uniform() < self.pulse_rate
                self.solution_fitness[i] = fitness
                self.pulse_rate_i[i] *= (1 - np.exp(-self.gamma * iteration))
```

```
import pandas as pd

# Load the dataset into a pandas DataFrame
data = pd.read_csv("../data/jm1.csv")

# Replace missing values with NaN
data = data.replace('?', np.nan)

# Fill missing values with a numerical value
data = data.fillna(-1)

# Separate the features and labels into separate variables
X = data.drop("defects", axis=1)
y = data["defects"].astype(int) # Convert boolean labels to integer
```

```
bat_algorithm = BatAlgorithm(population_size=10, frequency_min=0.0, frequency_max=2.0, pulse
```

```
bat_algorithm.init_bats(X, y)
```

```
for i in range(bat_algorithm.iterations):
    bat_algorithm.update_bats(i)
```

```
classifier = KNeighborsClassifier()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average="weighted")

print("Accuracy: ", accuracy)
print("F1 score: ", f1)
```

```
Accuracy:  0.7792406613594611
F1 score:  0.7465133011855375
```

4. Ant Colony Optimisation Algorithm

```
import numpy as np

class ACO:
    def __init__(self, num_ants=10, max_iter=100, pheromone_decay=0.5, alpha=1, beta=2):
        self.num_ants = num_ants
        self.max_iter = max_iter
        self.pheromone_decay = pheromone_decay
        self.alpha = alpha
        self.beta = beta

    def fit(self, X, y, clf):
        self.num_features = X.shape[1]
        self.pheromone = np.ones(self.num_features)
        self.feature_indices = np.arange(self.num_features)
        self.best_features = []
        self.best_score = -1

        for i in range(self.max_iter):
            ant_scores = []
            ant_features = []
            for j in range(self.num_ants):
                features = self.select_features()
                score = self.evaluate_features(features, X, y, clf)
                ant_scores.append(score)
                ant_features.append(features)
                if score > self.best_score:
                    self.best_score = score
                    self.best_features = features
            self.update_pheromone(ant_features, ant_scores)
        return self.best_features

    def select_features(self):
        features = []
        for i in range(self.num_features):
            pheromone_prob = self.pheromone[i] / np.sum(self.pheromone)
            rand_prob = np.random.rand()
            if rand_prob <= pheromone_prob:
                features.append(self.feature_indices[i])
        return features

    # def evaluate_features(self, features, X, y, clf):
    #     X_sub = X.iloc[:, features]
    #     X_train, X_test, y_train, y_test = train_test_split(X_sub, y, test_size=0.3, rando
    #     clf.fit(X_train, y_train)
    #     y_pred = clf.predict(X_test)
    #     score = f1_score(y_test, y_pred, average='weighted')
    #     return score

    def evaluate_features(self, features, X, y, clf):
        if len(features) == 0:
            return -1
        X_sub = X.iloc[:, features]
        X_train, X_test, y_train, y_test = train_test_split(X_sub, y, test_size=0.3, random_
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        score = f1_score(y_test, y_pred, average='weighted')
        return score

    def update_pheromone(self, ant_features, ant_scores):
        delta_pheromone = np.zeros(self.num_features)
        for features, score in zip(ant_features, ant_scores):
            for feature in features:
                delta_pheromone[feature] += score
        self.pheromone = self.pheromone * self.pheromone_decay + delta_pheromone
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from sklearn.neighbors import KNeighborsClassifier
# %pip install --no-binary aco-pants aco-pants
# from aco import ACO

# Load the dataset
data = pd.read_csv("../data/jm1.csv")

# Replace missing values with NaN
data = data.replace('?', np.nan)

# Drop rows with missing values
data = data.dropna()

# Separate the features from the target variable
X = data.drop('defects', axis=1)
y = data['defects']

# Convert categorical variables to numerical
X = pd.get_dummies(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, :-1], data.iloc[:, -1], test_size=0.3, ra

# Initialize the ACO algorithm with the parameters
aco = ACO(num_ants=10, max_iter=100, pheromone_decay=0.5, alpha=1, beta=2)

# Set the classifier to be used
clf = KNeighborsClassifier(n_neighbors=5)

# Fit the ACO algorithm on the training set
selected_features = aco.fit(X_train, y_train, clf)

# Train the classifier using the selected features
clf.fit(X_train.iloc[:, selected_features], y_train)

# Predict on the testing set using the trained classifier
y_pred = clf.predict(X_test.iloc[:, selected_features])

# Calculate the accuracy and F1 score of the classifier
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("F1 score:", f1)
```

```
Accuracy: 0.7858455882352942
F1 score: 0.7605301249081381
```

5. FireFly Algorithm

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, f1_score

# Define the Firefly Algorithm function
def firefly_algorithm(X_train, y_train, X_test, y_test, n_features, population_size=50, max_generations=10

    # Initialize the population
    population = np.random.randint(2, size=(population_size, n_features))

    # Iterate over generations
    for t in range(max_generations):

        # Evaluate the fitness of each individual
        fitness = np.zeros(population_size)
        for i in range(population_size):
            y_pred = np.dot(X_train, population[i])
            fitness[i] = accuracy_score(y_train, y_pred.round())

        # Sort the population by fitness
        indices = np.argsort(fitness)[::-1]
        population = population[indices]
        fitness = fitness[indices]
```

```
        # Iterate over fireflies
        for i in range(population_size):
            for j in range(population_size):
                if fitness[j] > fitness[i]:
                    r = np.sqrt(np.sum((population[i]-population[j])**2))
                    beta_ = beta * np.exp(-gamma * r**2)
                    population[i] = population[i].astype(float)
                    population[j] = population[j].astype(float)
                    population = population.astype(float)
                    population[i] += alpha * beta_ * (population[j] - population[i]) + np.random.uniform(-
                    # population[i] += alpha * beta_ * (population[j] - population[i]) + np.random.uniform

        # Apply mutation
        for i in range(population_size):
            for j in range(n_features):
                if np.random.rand() < 0.01:
                    population[i,j] = 1 - population[i,j]

    # Find the best individual and evaluate its performance on the test set
    fitness = np.zeros(population_size)
    for i in range(population_size):
        y_pred = np.dot(X_train, population[i])
        fitness[i] = accuracy_score(y_train, y_pred.round())
    best_index = np.argmax(fitness)
    y_pred = np.dot(X_test, population[best_index])
    test_accuracy = accuracy_score(y_test, y_pred.round())
    test_f1_score = f1_score(y_test, y_pred.round(), average='weighted')

    return test_accuracy, test_f1_score

# Load the dataset
data = pd.read_csv("../data/jm1.csv")

# Replace missing values with NaN
data = data.replace('?', np.nan)

# Drop rows with missing values
data = data.dropna()

# Split the dataset into training and test sets
train_data = data.sample(frac=0.8, random_state=1)
test_data = data.drop(train_data.index)
X_train = train_data.drop(columns=["defects"]).values.astype(float)
y_train = train_data["defects"].values
X_test = test_data.drop(columns=["defects"]).values.astype(float)
y_test = test_data["defects"].values

# Run the Firefly Algorithm
n_features = X_train.shape[1]
test_accuracy, test_f1_score = firefly_algorithm(X_train, y_train, X_test, y_test, n_features)

print("Test accuracy: {:.2f}".format(test_accuracy))
print("Test F1 score: {:.2f}".format(test_f1_score))
```

```
Test accuracy: 0.90
Test F1 score: 0.88
```

6. Genetic Algorithm

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, f1_score

# Define the Genetic Algorithm function
def genetic_algorithm(X_train, y_train, X_test, y_test, population_size=20, max_generations=100, mutation_

    # Initialize the population
    n_features = X_train.shape[1]
    # population = np.random.rand(population_size, n_features)
    population = np.random.randint(2, size=(population_size, n_features))

    # Iterate over generations
    for t in range(max_generations):
```

```
        # Evaluate the fitness of each individual
        fitness = np.zeros(population_size)
        for i in range(population_size):
            y_pred = np.dot(X_train, population[i])
            # y_pred = (np.dot(X_train, population[i]) > 0.5).astype(int)
            fitness[i] = accuracy_score(y_train, y_pred.round())

        # Select parents for reproduction
        parents = np.zeros((population_size//2, n_features))
        for i in range(population_size//2):
            # indices = np.random.choice(population_size, size=2, replace=False, p=fitness/np.sum(fitness)
            fitness[fitness==0] = 1e-10
            indices = np.random.choice(population_size, size=2, replace=False, p=fitness/np.sum(fitness))
            parents[i] = population[indices[0]]

        # Create offspring by crossover
        offspring = np.zeros((population_size//2, n_features))
        for i in range(0, population_size//2, 2):
            alpha = np.random.rand()
            offspring[i] = alpha*parents[i] + (1-alpha)*parents[i+1]
            offspring[i+1] = (1-alpha)*parents[i] + alpha*parents[i+1]

        # Mutate some of the offspring
        for i in range(population_size//2):
            if np.random.rand() < mutation_rate:
                j = np.random.randint(n_features)
                offspring[i,j] = np.random.rand()

        # Combine the parents and offspring to form the next generation
        population[:population_size//2] = parents
        population[population_size//2:] = offspring

    # Find the best individual and evaluate its performance on the test set
    fitness = np.zeros(population_size)
    for i in range(population_size):
        y_pred = np.dot(X_train, population[i])
        fitness[i] = accuracy_score(y_train, y_pred.round())
    best_index = np.argmax(fitness)
    y_pred = np.dot(X_test, population[best_index])
    test_accuracy = accuracy_score(y_test, y_pred.round())
    test_f1_score = f1_score(y_test, y_pred.round(), average='weighted')

    return test_accuracy, test_f1_score

    # return test_accuracy

# Load the dataset
data = pd.read_csv("../data/jm1.csv")

# Replace missing values with NaN
data = data.replace('?', np.nan)
```

```
# Drop rows with missing values
data = data.dropna()

# Split the dataset into training and test sets
train_data = data.sample(frac=0.8, random_state=1)
test_data = data.drop(train_data.index)
X_train = train_data.drop(columns=["defects"]).values.astype(float)
y_train = train_data["defects"].values
X_test = test_data.drop(columns=["defects"]).values.astype(float)
y_test = test_data["defects"].values

# Run the Genetic Algorithm
# print(X_train.shape)
# test_accuracy = genetic_algorithm(X_train, y_train, X_test, y_test)

# # print("Accuracy: {:.2f}".format(test_accuracy))
# # f1 = f1_score(y_test, y_pred, average='weighted')

# # print("Accuracy:", accuracy)
# # print("F1 score:", f1)

test_accuracy, test_f1_score = genetic_algorithm(X_train, y_train, X_test, y_test)
print("Accuracy: {:.2f}".format(test_accuracy))
print("F1 score: {:.2f}".format(test_f1_score))
```

```
Accuracy: 0.81
F1 score: 0.72
```

REFERENCES

[1] The Institute of Electrical and Electronics Engineers. “29119 -1-2013 - Software and systems engineering —Software testing.” IEEE Standards Association. .

[2] International Organization for Standardization. “What Is a Standard?” ISO Standards.

[3] I Gondra, Applying machine learning to software fault-proneness prediction, Journal of Systems and Software, 2008

[4] P. Oman and J. Hagemeister, “Construction and testing of polynomials predicting software maintainability,” J. Syst. Softw., vol. 24, no. 3, pp. 251–266, Mar. 1994.

[5] Naik K, Tripathy P. Software Testing and Quality Assurance. Theory and Practice. Wiley, 2008, 616p.

[6] Myers G, Badgett T, Sandler C. The Art of Software Testing. 3rd Edition. Hoboken, NJ: J. Wiley & Sons; 2014.

[7] Spillner A, Linz T, Schaefer H. Software Testing Foundations. 4th Edition.. Santa Barbara: Rocky Nook Inc.; 2014.

[8] Meyer B. Seven Principles of Software Testing. Computer 2008, August:99-101.

[9] Autili M, Di Salle A, Gallo F, Perucci A, and Tivoli M., Biological Immunity and Software Resilience: Two Faces of the Same Coin?, in A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, DOI:10.1007/978-3-319-23129-71, 1–15, 2015.

[10] Madsen H, Thyregod P, Burtschy B, Albeanu G, Popentiu F. A Fuzzy Logic Approach to Software Testing and Debugging. In: Guedes Soares, Zio E, editors. Safety and Reliability for Managing Risk , London:Taylor & Francisc Group; 2006, p. 1435-1442.

[11] C. Catal and B. Diri, “A systematic review of software fault prediction studies,” Expert Syst. Appl., vol. 36, no. 4, pp. 7346–7354, May 2009.

[12] V. U. B. CHALLAGULLA, F. B. BASTANI, I.-L. YEN, and R. A. PAUL, “EMPIRICAL ASSESSMENT OF MACHINE LEARNING BASED SOFTWARE DEFECT PREDICTION TECHNIQUES,” Int. J. Artif. Intell. Tools, vol. 17, no. 02, pp. 389–400, Apr. 2008.

[13] Madsen H, Thyregod P, Burtschy B, Albeanu G, Popentiu F. On using soft computing techniques in software reliability engineering. International Journal of Reliability, Quality, and Safety Engineering 2006;13(1):1–12

[14] Sharma C, Sabharwal S, Sibal R. A Survey on Software Testing Techniques Using Genetic Algorithm. International Journal of Computer Science Issues 2013;10(1):381-393.

[15] H. Tanwar and M. Kakkar, "A Review of Software Defect Prediction Models," Springer, Singapore, 2019, pp. 89–97 - 12th Sept 2019

[16] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Appl. Soft Comput., vol. 27, pp. 504–518, Feb. 2015 - Feb 2015

[17] Tim Menzies, Justin DiStefano, Andres Orrego, Robert (Mike) Chapman, "Assessing Predictors of Software Defects" - Jan 2004

[18] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in Proceedings of the 6th International Conference on Predictive Models in Software Engineering - PROMISE '10, 2010, p. 1 - 12th Sept 2010

[19] Malhotra, R.; Jain, A. Software fault prediction for object-oriented systems: A systematic literature review. ACMSIGSOFT Softw. Eng. 2011, 36, 1–6.

[20] Malhotra, R. A systematic review of machine learning techniques for software fault prediction. Appl. Soft Comput. 2015, 27,504–518.

[21] Radjenovic, D.; Heriko, M. Software fault prediction metrics: A systematic literature review. Inf. Softw. Technol. 2013, 55,1397–1418.

[22] Misirli, A.T.; Bener, A.B. A mapping study on Bayesian networks for software quality prediction. In Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, Hyderabad, India, 3 June 2014; pp. 7–11.

[23] Murillo-Morera, J.; Quesada-López, C.; Jenkins, M. Software Fault Prediction: A Systematic Mapping Study; CIbSE: London, UK, 2015; p. 446.

[24] Özakıncı, R.; Tarhan, A. Early software defect prediction: A systematic map and review. J. Syst. Softw. 2018, 144, 216–239.