A Study on Unsupervised Machine Learning Algorithms for detection of Phishing Attacks

Thesis submitted towards partial fulfilment
of the requirements for the degree of

Master of Technology in IT (Courseware Engineering)

Submitted by
Roni Das

EXAMINATION ROLL NO. M4CWE24001
UNIVERSITY REGISTRATION NO. 163769 of 2022-23

Under the guidance of
Dr SASWATI MUKHERJEE

School of Education Technology
Jadavpur University

Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata-700032
India
2024

M.Tech in IT (Courseware Engineering)
Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled "**A Study on Unsupervised Machine Learning Algorithms for detection of Phishing Attacks using URL Parameters**" is a bona fide work carried out by **Roni Das** under our supervision and guidance for partial fulfilment of the requirements for the degree of **Master of Technology in IT (Courseware Engineering)** in **School of Education Technology**, during the academic session 2023-2024.

Dr. SASWATI MUKHERJEE
SUPERVISOR
School of Education Technology
Jadavpur University,
Kolkata-700 032

DIRECTOR
School of Education Technology
Jadavpur University,
Kolkata-700 032

DEAN - FISLM
Jadavpur University,
Kolkata-700 032

M.Tech in IT (Courseware Engineering)
Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

CERTIFICATE OF APPROVAL **

This foregoing thesis is hereby approved as a credible study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned does not endorse or approve any statement made or opinion expressed or conclusion drawn therein but approves the thesis only for the purpose for which it has been submitted.

Committee of final examination for evaluation of the Thesis

** Only in case the thesis is approved.

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of his **Master of Technology in IT (Courseware Engineering)** studies.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by this rule and conduct, I have fully cited and referenced all materials and results that are not original to this work.

NAME	:	Roni Das
EXAMINATION ROLL NUMBER	:	M4CWE24001
REGISTRATION NUMBER	:	163769 of 2022-23
THESIS TITLE	:	A Study on Unsupervised Machine Learning Algorithms for detection of Phishing Attacks using URL Parameters.

SIGNATURE: DATE:

Acknowledgment

I feel fortunate while presenting this dissertation at **the School of Education Technology, Jadavpur University, Kolkata,** in the partial fulfilment of the requirement for the degree of **M. Tech in IT (Courseware Engineering).**

I hereby take this opportunity to show my gratitude to my mentor, **Dr Saswati Mukherjee,** who has guided and helped me with all possible suggestions, support, advice, and constructive criticism, along with illuminating views on different issues of this dissertation, which helped me throughout my work.

I want to thank **Prof. Dr. Matangini Chattopadhyay, Director of the School of Education Technology,** for her timely encouragement, support, and advice. I would also like to thank **Mr. Joydeep Mukherjee** for his constant support during my entire course of work.

My thanks and appreciation go to my classmates from M.Tech in IT (Courseware Engineering) and Master in Multimedia Development.

I wish to thank all the departmental support staff and everyone else who has contributed to this dissertation.

Finally, I would like to express my special gratitude to my parents, who have invariably sacrificed, supported me, and helped me achieve this height.

Date:
Place: Kolkata

Roni Das
Examination Roll Number: M4CWE24001
M.Tech in IT (Courseware Engineering)
School of Education Technology
Jadavpur University
Kolkata:70003

Contents

List of Figures

Executive Summary

This dissertation presents a novel approach to identifying phishing attacks by leveraging unsupervised machine-learning techniques through URL analysis. Unlike most existing study, which relies on supervised learning methods that require extensive training on labelled data, this study introduces a method that enables the detection of phishing attempts without prior knowledge of malicious URLs.

The present work applied k-means and Gaussian Mixture Models (GMM) to detect phishing attacks. It analysed URL components to uncover parameters indicative of phishing, employed nature-inspired and genetic algorithms for feature selection, and assessed the performance of the clustering models using the silhouette score. The model's effectiveness was tested with an external data set to ensure robustness.
The methodology begins with URL preprocessing and applies four feature selection algorithms: Ant Colony, Dragonfly, Particle Swarm Optimization (PSO), and Binary PSO. The best feature subset identified is then used to perform clustering with k-means and GMM. The k-means algorithm, paired with the PSO-selected features, achieved the highest silhouette score, signifying the most accurate clustering. The findings underscore the potential of unsupervised learning in detecting phishing attacks.

This work can be expanded by incorporating features like behavioural data to enhance phishing detection accuracy. Integrating unsupervised methods with semi-supervised learning could improve performance in scenarios with limited labelled data. Developing real-time detection systems based on these techniques would allow immediate threat response. Exploring the model's adaptability to other domains, like spam detection, could further demonstrate its versatility. Finally, creating user-friendly tools that implement these methods could make advanced phishing detection more accessible to cybersecurity professionals.

Introduction

1. Overview

Phishing is a cyber-attack that targets naive online users, tricking them into revealing sensitive information such as username, password, social security number, credit card number etc. Attackers fool Internet users by masking web pages as trustworthy or legitimate pages to retrieve personal information. There are many anti-phishing solutions, such as blacklisting or whitelisting, and heuristic and visual similarity-based methods proposed to date. However, online users are still trapped in revealing sensitive information on phishing websites [1]. Researchers have integrated machine learning techniques into network security, yielding promising results in enhancing cybersecurity measures. Phishing attacks pose significant threats to individuals and organisations worldwide, with attackers using deceptive methods to obtain sensitive information. Detecting phishing attacks and URLs has become challenging due to the dynamic nature of web content. Traditional supervised machine-learning approaches for phishing URL detection rely on labelled datasets, which can be resource-intensive and impractical due to the rapid proliferation of new phishing URLs [2]. In response to these challenges, the present work focuses on exploring and applying unsupervised machine-learning techniques for detecting phishing URLs. Unsupervised learning offers a promising alternative by enabling the detection of patterns and anomalies in data without the need for labelled examples. By leveraging URL data's inherent structure and characteristics, unsupervised models aim to identify suspicious URLs based on their deviation from normal web behaviour.

1.2 Problem Statement

A Study on Unsupervised Machine Learning Algorithms for detection of Phishing Attacks.

1.3 Objectives

- To detect phishing attacks using unsupervised algorithms, namely, k-means and GMM.
- To analyses the URL parameters responsible for the phishing attack.
- To select the relevant features using nature-inspired and genetic algorithms for clustering the normal and the malicious URLs.
- To evaluate the clustering performance of k-means and GMM algorithms based on the silhouette score.
- To validate the robustness of the model fit using an external dataset.

2.Background Concept

Phishing attack detection by analyzing URL components involves scrutinizing various URL elements to identify suspicious patterns indicative of phishing attempts. Key components examined include the domain name, path, query parameters, and unusual characters or structures deviating from legitimate URLs. To enhance detection accuracy, feature selection methods such as Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Dragonfly Algorithm, and Binary Particle Swarm Optimization (BPSO) are employed to identify the most relevant features from the URL data. These selected feature sets are then used to train clustering models, specifically K-means and Gaussian Mixture Model (GMM), to group similar URLs. The performance of these clustering models is evaluated using the silhouette score, which measures how well the URLs are grouped. By comparing the silhouette scores, the effectiveness of each feature selection technique in providing the best clustering results is determined, aiding in the identification of the most robust method for phishing detection.

2.1. Feature Selection Methods

Feature selection is a vital aspect of machine learning. Feature selection is applied to reduce the number of features in many applications where data has hundreds or thousands of features. Existing feature selection methods mainly focus on finding relevant features. By carefully choosing the appropriate set of features, the performance of clustering models can be significantly improved [3]. This work explores four different methods of feature selection: the Ant Colony Optimization algorithm, Dragonfly algorithm, PSO and Binary PSO.

2.1.1 Ant Colony Optimization (ACO) Technique

Ant Colony Optimization (ACO) is a computational technique inspired by the foraging behaviour of ants in nature. It solves complex optimization problems, such as finding the shortest path between an ant colony and a food source. In this process, ants leave pheromone trails on their travel paths, which guide other ants to the food source. Over time, shorter paths are reinforced by accumulating more pheromones, while longer paths experience pheromone evaporation. In ACO, this behaviour is simulated using artificial 'ants' to find optimal solutions in a search space. These ants construct solutions as they move through possible states, depositing virtual pheromones that influence the paths of subsequent ants. The algorithm is refined iteratively, balancing exploring new paths with exploiting known good paths, making it particularly effective for problems like the travelling salesman, network routing, and scheduling. In Figure 1, the Ant Colony Optimization process is summarized, showing how

exploration identifies potential paths, optimization refines these paths, and the final step extracts the optimal solution.

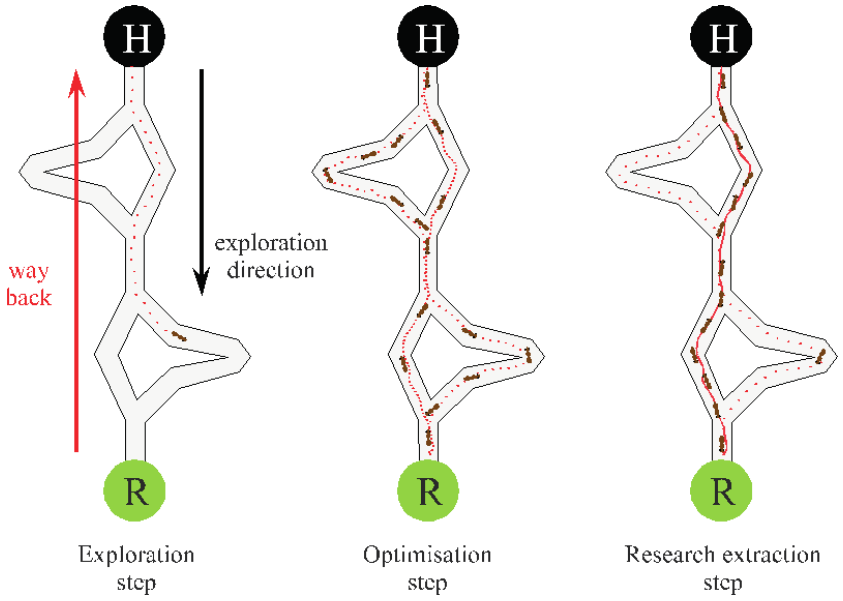

Figure 1: Ant colony optimizer

The adaptive and self-organizing nature of ACO allows for handling dynamic changes and efficiently finding near-optimal solutions. In the context of feature selection, ants iteratively choose features based on pheromone levels, which are updated to reflect the quality of selected features. This approach is particularly useful for selecting a subset of features with numerous attributes in datasets [4]. The Algorithm of Antcolony can be found in Algorithm 1

The ACO class is initialized with the list of features, the number of ants, the maximum number of iterations, and the initial pheromone value. This creates a pheromone matrix with the same length as the number of features, which is initialized to the initial pheromone level. Table 1 Presents the Hyperparameter of Antcolony algorithm and their description.

Table 1: Hyperparameters of Ant colony optimizer

Hyperparameter	**Description**
Number of Ants(n)	Number of ants used in each run of the algorithm. More ants can explore more solutions but take more time.
Number of Iterations	More iterations allow for a more thorough search for the best solution.
Pheromone Evaporation Rate: rho	How quickly the pheromone trail fades away. A slower rate means the path is remembered longer,

	while a faster rate encourages exploring new paths.
Alpha (α)	How much the ants follow the pheromone trail. Higher values mean ants are more likely to follow the trail.
Beta (β)	How much do the ants consider other factors like distance or cost when choosing a path? Higher values mean ants pay more attention to these factors.
Initial Pheromone Level	How much pheromone does each ant leave on its path? More pheromones can lead to quicker solutions but might miss the best one.

Algorithm 1: ANTCOLONY

```
Initialize pheromone levels on all paths
Set parameters:
    alpha (pheromone importance)
    beta (visibility importance)
    rho (evaporation rate)
    Q (pheromone deposit)
    number of ants
While stopping criteria not met do
     For each ant do
          Initialize tour for this ant
        While tour is not complete do
             Choose next city based on pheromone levels and visibility (using probabilities)
              Move to the next city and update tour
         End While
         Evaluate the tour (calculate tour length or cost)
          Update best tour if necessary
       End For

    Update pheromone levels
       For each path do
          Evaporate pheromone
          Deposit new pheromone based on the best tour(s)
       End For
End While
Return the best tour fo
```

2.1.2 DRAGONFLY ALGORITHM

The Dragonfly Algorithm (DA) is an innovative nature-inspired optimization technique that emulates dragonflies' static and dynamic swarming behaviour in their natural environment. The algorithm effectively navigates and optimize complex search spaces by simulating these behaviour. DA strikes a balance between exploration (searching new areas) and exploitation (refining known areas) through the coordinated movements of artificial dragonflies, which are guided by principles such as alignment (synchronization with others), cohesion (tendency to stay together), separation (avoiding crowding), attraction to food sources, and avoidance of threats. This approach has demonstrated superior performance to many traditional meta-heuristic optimization methods [5]. The Dragonfly algorithm is represented in Figure 2, showcasing its core behavioral mechanisms: separation, alignment, cohesion, attraction to food sources, and distraction from enemies. These components play a crucial role in steering the optimization process effectively.

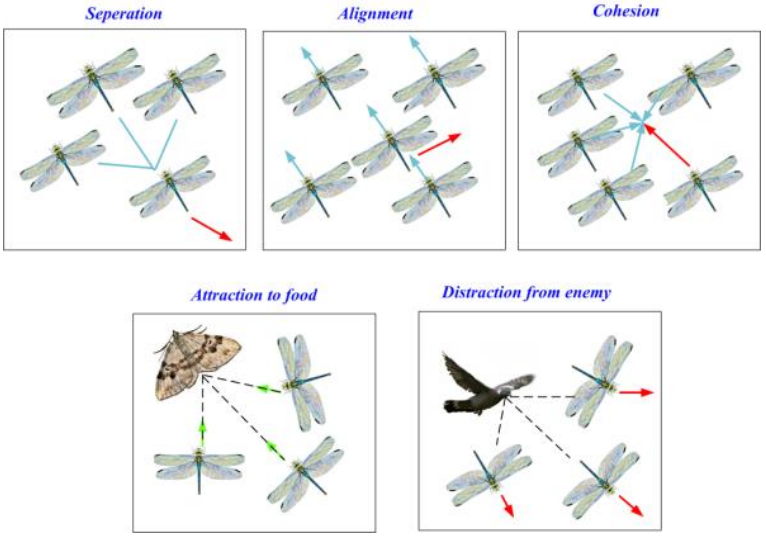

Figure 2: Dragonfly Algorithm

Initialization: The Dragonfly Algorithm randomly selects an initial set of features from the dataset.

Evaluation: It evaluates the performance of the selected features using a predefined evaluation criterion, such as accuracy or error rate, on a validation dataset.

Update: Based on the evaluation results, the algorithm updates the selected features using its optimisation strategy, which involves adjusting the feature subset to improve performance.

Iteration: The algorithm repeats the evaluation and update steps for several iterations or

until a stopping criterion is met.

Optimization: The algorithm continuously optimizes the feature subset, gradually improving its performance based on the evaluation criterion.

Final Selection: Once the optimization process is complete, the algorithm selects the final subset of features that maximizes performance according to the evaluation criterion.

The Algorithm of Dragonfly can be found in Algorithm 2,

Table 2 Presents the Hyperparameter of Dragonfly algorithm and their description.

Table 2: Hyperparameters of Dragonfly Algorithm

Hyperparameter	**Description**
Number of Dragonflies	Determines the size of the population, impacting exploration capability.
Number of Iterations	Controls how often the algorithm runs, affecting the thoroughness of solution search.
Inertia Weight (w)	Controls the influence of the previous velocity on the current velocity of each dragonfly. It helps balance exploration and exploitation.
Step size;x	Controls the step size or how far dragonflies move in each iteration. It affects the convergence speed and accuracy.
Absorption_coffeicient	The absorption coefficient influences how dragonflies respond to external factors such as threats or constraints

Algorithm 2: Dragonfly

01. Initialize the population of dragonflies (positions and velocities)
02. Initialize the step size (ΔX), separation weight (w_s), alignment weight (w_a), cohesion weight (w_c),
03. Initialize food factor weight (w_f), enemy factor weight (w_e), and other algorithm-specific parameters
04. Evaluate the fitness of each dragonfly in the population
05. **While** stopping criteria not met do
06. **For** each dragonfly do
07. Calculate the separation (S), alignment (A), and cohesion (C) components based on neighboring dragonflies
08. Calculate the attraction towards food source (F) and distraction from enemies (E)
09. Update velocity of the dragonfly using the following formula:
10. velocity = w_s * S + w_a * A + w_c * C + w_f * F + w_e * E
11. Update the position of the dragonfly using:
12. position = position + velocity
13. Apply boundary conditions if the dragonfly goes out of bounds
14. Evaluate the fitness of the updated dragonfly position
15. Update the best position found by the dragonfly if the new position is better
16. **End** For
17. Adjust the step size (ΔX) and other dynamic parameters as necessary
18. **End** While
19. Return the best position found by any dragonfly as the solution

2.1.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a nature-inspired optimization technique based on the social behaviour of birds flocking or fish schooling. It operates with a population of candidate solutions, particles, that move through the search space by adjusting their velocities based on individual and collective experiences. Each particle updates its position, which is influenced by its own best-known position and the best-known positions of its neighbours, effectively balancing exploration and exploitation of the search space. This dynamic adjustment helps the swarm efficiently converge towards optimal or near-optimal solutions [6].

PSO is particularly valued for its simplicity, ease of implementation, and robustness in handling non-linear, multi-dimensional, and complex optimization problems. It has been successfully applied across various domains, including machine learning, engineering design, and economic modelling. The algorithm's ability to find high-quality solutions with relatively low computational cost makes it popular for diverse optimization tasks.

The description of the algorithm is as follows.

Starting Point: PSO begins by randomly picking a group of potential feature subsets.

Moving Around: Each particle adjusts its position (feature selection) based on two things: It's the previous best choice (personal best). The best choice is made by any particle in the group (global best). It does this by considering how much it should move towards these better choices.

Checking Fitness: After moving, each particle checks how good its new feature subset is using a fitness measure. This tells us how well the features predict or classify.

Updating Best Choices: If a particle finds a better solution (higher fitness) than before, it remembers it as its personal best. If any particle in the group finds an even better solution, then it is the new global best.

Repeating: The above steps are repeated for a set number of rounds or until there's no more improvement.

Final Decision: The feature subset with the highest fitness (either from a single particle or from the group) is chosen as the final selection. So, PSO helps find the best combination of features by having particles move around in the feature space, learning from their own experience and the group's best choices. Table 3 Presents the Hyperparameter of PSO algorithm and their description. The Algorithm of PSO can be found in Algorithm 3.

Table 3: Hyperparameters of PSO

Hyperparameter	Description
Cognitive Coefficient (c1)	The number of trees in the forest
Social Coefficient (c2)	The maximum depth of the individual trees
Inertia Weight (ω)	The minimum number of samples required to split an internal node
Best known position of particle(P)	minimum number of samples required to be at a leaf node
Current velocity of particle(V)	velocity of the particles
Number of Iterations	The maximum number of iterations the algorithm will run
Swarm Size	The number of particles in the swarm.

Algorithm 3: PSO

01. Initialize:

02. Initialize the swarm of particles with random positions and velocities.

03. For each particle, set the initial position as its personal best (pBest).

04. Identify the global best (gBest) position among all particles based on the objective function.

05. Repeat until convergence (or max iterations):

06. **For** each particle:

07. Update the particle's velocity:

08. velocity = nertia_weight * current_velocity

+ cognitive_constant * rand()

* (pBest_position -

current_position)

```
                + social_constant * rand() *
                (gBest_position- current_position)
    Update the particle's position:
        current_position = current_position + velocity
    Evaluate the particle's fitness based on
    the objective function.
    Update personal best (pBest) if
    the current fitness is  better:
        If current_fitness < pBest_fitness:
            pBest_position = current_position
            pBest_fitness = current_fitness
Update the global best (gBest) if any particle's
personal best is better:
    If any pBest_fitness < gBest_fitness:
        gBest_position = that_particle's_pBest_position
        gBest_fitness = that_particle's_pBest_fitness
Return the best position (gBest) and the corresponding
fitness value.
```

2.1.4 Binary Particle Swarm Optimization

Binary Particle Swarm Optimization (BPSO) is an adaptation of the standard Particle Swarm Optimization (PSO) for solving binary (discrete) optimization problems. In BPSO, each particle represents a solution in a binary format, where each dimension can take a value of either 0 or 1. The algorithm updates the velocity of each particle based on its own best-known position and the best-known positions of its neighbours, similar to standard PSO. To convert the continuous velocity values to binary values, BPSO uses a sigmoid function, which outputs probability. This probability determines whether each dimension of a

particle's position should be set to 0 or 1. The particle's position is then updated by comparing this probability to a random number, effectively flipping the bits accordingly. BPSO is particularly useful for discrete optimization problems, where solutions must be represented in a binary format, such as feature selection in machine learning and network design problems [7].

Initialization: BPSO starts by initializing a population of binary strings, each representing a possible solution. In feature selection, each bit in the binary string corresponds to a feature being selected (1) or not selected (0).

Evaluation: The fitness of each solution (particle) in the population is evaluated based on a fitness function. In feature selection, this function typically measures the quality of the subset of features selected.

Updating Velocity and Position: BPSO then updates each particle's velocity and position based on its current position, the best position it has achieved so far (personal best), and the best position achieved by any particle in the population (global best). The velocity determines the direction and magnitude of movement for each particle.

Movement: Each particle adjusts its position according to its velocity, possibly flipping bits in its binary string representation to explore the search space.

Update Personal and Global Best: After moving, each particle updates its personal best position if the new position has a better fitness value. The global best position is updated if any particle achieves a better fitness value than the current best.

Termination: The algorithm terminates when a stopping criterion is met, such as reaching a maximum number of iterations or when the improvement in fitness becomes negligible.

Extraction: Finally, the best solution found (global best) represents the selected subset of features. BPSO is particularly effective for feature selection because it explores the search space efficiently, quickly converging towards promising solutions. By iteratively updating the positions of particles based on their own experience and the experiences of the entire swarm, BPSO can effectively navigate the high-dimensional feature space to find a subset that optimizes the specified fitness function. The Algorithm of BPSO can be found in Algorithm 4.

Algorithm 4: BPSO

01. Initialize:

02. Initialize the swarm of particles with random binary positions (0s and 1s) and velocities.

03. For each particle, set the initial position as its personal best (pBest).

04. Identify the global best (gBest) position among all particles based on the objective function.

05. Repeat until convergence (or max iterations):

06. **For** each particle:

07. **For** each dimension in the particle's position:

08. Update the particle's velocity:

09. velocity[d] = inertia_weight * current_velocity[d] + cognitive_constant * rand() * (pBest_position[d] - current_position[d]) + social_constant * rand() * (gBest_position[d] - current_position[d])

10. Apply the sigmoid function to the velocity:

11. probability[d] = 1 / (1 + exp(-velocity[d]))

12. Update the particle's position using the probability:

13. **If** rand() < probability[d]:

14. current_position[d] = 1

15. **Else**:

16. current_position[d] = 0

17. Evaluate the particle's fitness based on the objective function.

20. Update personal best (pBest) if the current fitness is better:

21. **If** current_fitness < pBest_fitness:

22. pBest_position = current_position

23. pBest_fitness = current_fitness

24. Update the global best (gBest) if any particle's personal best is better:

25. **If** any pBest_fitness < gBest_fitness:

26. gBest_position = that_particle's_pBest_position

27. gBest_fitness = that_particle's_pBest_fitness

28. **Return** the best position (gBest) and the corresponding fitness value

2.2 Clustering Method

Nowadays, many industries deal with very large data sets of different types. Manually processing all that information can be time-consuming and might not even add value in the long term. Many strategies, from simple automation to machine learning techniques, are being applied for a better return on investment. Clustering is an unsupervised machine learning method in which the model tries to group similar data points into clusters based on their inherent characteristics without predefined labels. In clustering, the model is trained to identify patterns and similarities within the data, enabling the discovery of natural groupings and structures. For instance, an algorithm can learn to group customers into different segments based on their purchasing behaviour, as illustrated below. Various clustering algorithms, such as K-means, hierarchical clustering, and DBSCAN, can be used, each with its approach to defining and discovering clusters. K-means is popular for dividing data into a predetermined number of clusters.

2.2.1 K-means

K-Means is a popular clustering algorithm used to partition a dataset into distinct groups or clusters based on feature similarity. It starts by randomly selecting K centroids, which act as the initial centers of the clusters. Each data point in the dataset is then assigned to the nearest centroid, forming K clusters. After the initial assignment, the centroids are recalculated as the mean of all data points within their respective clusters. Data points are

reassigned to the nearest centroid based on the updated centroid positions. This process of recalculating centroids and reassigning data points is repeated iteratively until the centroids no longer change significantly, indicating that the clusters have stabilized. The final outcome is K distinct clusters, where each cluster contains data points that are more similar to each other than to those in other clusters. [8]. The Algorithm of K-Means can be found in Algorithm 5.

You keep repeating these steps—adjusting the jars' positions and reassigning marbles—until the jars stop moving significantly or a predefined number of iterations is reached.

Algorithm 5: K-Means

01: Initialize k centroids randomly from the data points.

02: Repeat until convergence:

03: a. Assignment step:

04: For each data point, calculate the distance to each centroid.

05: ii. Assign each data point to the nearest centroid.

06: b. Update step:

07: For each cluster, calculate the new centroid by averaging the coordinates of all data points assigned to it.

08: c. Check for convergence:

09: If centroids do not change or the change is minimal, stop the algorithm.

10: Return the final centroids and cluster assignments.

The final positions of the jars and the marbles they contain represent the clusters found by the algorithm. However, just like in real life, sometimes you might end up with suboptimal groups if the initial placement of the jars was unlucky or if the data doesn't naturally fall into distinct clusters. Despite its simplicity, K-Means is widely used for its efficiency and effectiveness in many applications, such as market segmentation, image compression, and

data preprocessing. As shown in Figure 3, k-means clustering effectively transforms similar data points into distinct cluster groups, demonstrating the separation achieved after the clustering process.

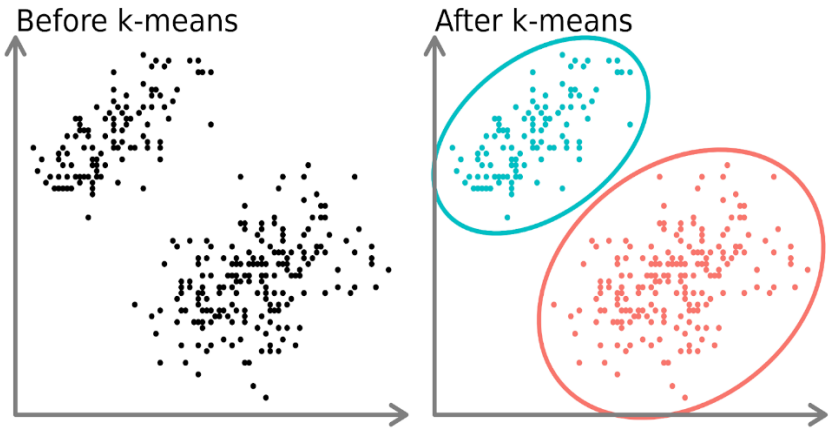

Figure 3: K-Means clustering

2.2.2 Gaussian Mixture Model (GMM)

A Gaussian Mixture Model (GMM) is a probabilistic model used for clustering by representing data as a mixture of multiple Gaussian distributions. Each Gaussian distribution corresponds to a cluster and is defined by its mean, covariance, and weight. GMM uses the Expectation-Maximization (EM) algorithm to iteratively update these parameters, maximizing the likelihood of the data. Unlike k-means, GMM performs soft clustering, assigning probabilities to data points for belonging to each cluster. This allows GMM to handle clusters of different shapes, sizes, and overlapping regions. The number of Gaussian components is chosen using criteria like BIC or AIC. GMM is useful in applications like clustering, anomaly detection, and density estimation.

GMM works by modelling data as a mixture of Gaussian distributions, each representing a cluster. It uses the Expectation-Maximization (EM) algorithm to iteratively update the parameters (mean, covariance, weight) of these distributions by calculating the probabilities of data points belonging to each cluster. The process repeats until the parameters converge, resulting in a model that captures complex and overlapping cluster structures [9].

As in the candy example, GMM iteratively adjusts its parameters to best explain the data. It assumes that the data is generated from a mixture of several Gaussian distributions, each representing a different cluster or group.

GMM is useful for tasks like image segmentation, where pixels with similar characteristics must be grouped together or for identifying underlying patterns in complex datasets.

However, like any model, GMM has limitations, such as assuming that the data is normally distributed and specifying the number of components (candy types) beforehand. Figure 4 shows GMM clustering with clusters differentiated by their standard deviations. The plot highlights how varying standard deviations lead to distinct cluster shapes and spreads. The Algorithm of GMM can be found in Algorithm 6.

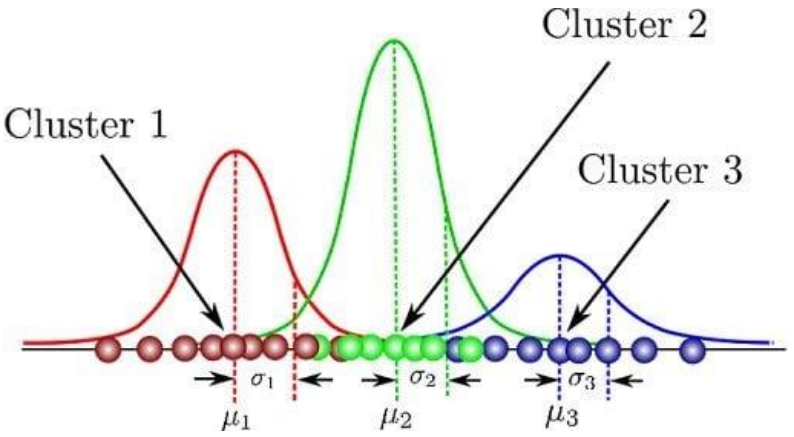

Figure 4: Gaussian Mixture Model

Algorithm 6: GMM

```
# Initialize the parameters
K = number_of_components          # Number of Gaussian components
mu = initialize_means(K)          # Means (μ) for each component
sigma = initialize_covariances(K) # Covariance matrices (Σ) for
                                                    each component
```

```
pi = initialize_mixing_coeffs(K)  # Mixing coefficients (π) for each component
converged = False                 # Convergence flag

# Repeat until convergence
while not converged:

    # E-Step: Calculate the responsibilities (γ_ik)
    for i in range(N):            # Loop over each data point x_i
        for k in range(K):        # Loop over each component k
            # Compute the probability that x_i belongs to component k
            prob_k = pi[k] * gaussian_pdf(x[i], mu[k], sigma[k])

            # Calculate the responsibility γ_ik
            responsibilities[i, k] = prob_k / sum(pi[j] *
                gaussian_pdf(x[i], mu[j], sigma[j])
                 for j in range(K))

    # M-Step: Update the parameters (μ, Σ, π)
    for k in range(K):                # Loop over each component k
        # Update the mixing coefficient π_k
        N_k = sum(responsibilities[i, k] for i in range(N))
        pi[k] = N_k / N

        # Update the mean μ_k
        mu[k] = sum(responsibilities[i, k] * x[i] for
              i in range(N)) / N_k

        # Update the covariance matrix Σ_k
        sigma[k] = sum(
            responsibilities[i, k] * np.outer(x[i] - mu[k],
                    x[i] - mu[k])
            for i in range(N)


        ) / N_k

    # Check for convergence (based on log-likelihood or parameter change)
    if log_likelihood_change < threshold or parameter_change < threshold:
        converged = True
```

3. Literature Survey

Bahnsen et al. [10] used the application of machine learning models for predicting phishing sites using URLs as input. They compared two methods: a feature-engineering approach combined with a random forest classifier and a novel approach utilizing recurrent neural networks (RNNs). Their study found that the RNN-based method achieved an impressive accuracy rate of 98.7%, surpassing the random forest approach by 5%. This result highlights the RNN method as a highly effective, scalable, and fast-acting detection system that does not rely on manual feature creation or complete content analysis. This work demonstrates the potential of RNNs for proactive phishing site detection, offering significant improvements in accuracy and efficiency.

Li et al. [11] explored the critical role of feature selection as a data preprocessing strategy, especially for high-dimensional data in data mining and machine learning. The main goals of feature selection include simplifying models, enhancing data-mining performance, and ensuring clean, interpretable data. The survey highlights the challenges and opportunities presented by the proliferation of big data. It provides a comprehensive overview of recent advances in feature selection, categorizing algorithms into four main groups: similarity-based, information-theoretical-based, sparse-learning-based, and statistical-based methods. The survey also introduces an open-source repository for feature selection algorithms, facilitating research and evaluation in this field.

Al-Ani et al. [12] proposed a novel approach to this problem using Ant Colony Optimization (ACO), a metaheuristic inspired by ants' behaviour in finding the shortest paths to food. The ACO algorithm optimizes feature selection by balancing local heuristics with knowledge from previous iterations. Applied to two classification problems, the method was tested using five baseline feature vectors input to an Artificial Neural Network (ANN). The study involved 71,354 training patterns and 23,785 testing patterns. The classification accuracies achieved were 76.17%, 76.04%, 74.06%, 75.23%, and 89.39%. The results demonstrated the ACO-based method's potential, with one feature vector significantly outperforming the others. This approach shows promise in enhancing the efficiency and effectiveness of feature selection in pattern classification.

Sekhar et al. [13] proposed using the Dragonfly Algorithm (DFA) for feature selection in skin disease classification. DFA was applied to identify key features for illness categorization and paired with CNN models, including VGG19 and EfficientNet-B2, for classification tasks. The algorithm evaluated feature sets by measuring the accuracy of classifiers on a training dataset, ensuring precise selection. Experimental results showed

DFA's high precision and minimal loss. Two CNN models, based on EfficientNet-B2 and VGG19, were trained on DermNet NZ and ISIC 2019 datasets. These models achieved an average accuracy of 88.5% and a loss of 0.0003 across eight skin diseases. The study demonstrated deep learning's potential to classify skin conditions with near-human accuracy. It also highlighted the potential for large-scale, real-time skin disease diagnosis, enhancing healthcare practices and patient outcomes [11].

Firpi et al. [14] used a feature extraction method based on Particle Swarm Optimization (PSO) to monitor brain activity and identify cognitive states and task intensity. This approach aims to develop a pattern recognition system that classifies cognitive states, enabling workload redistribution among subjects. The system utilises multiple features from different domains, with PSO employed for feature selection. Classification is performed using the k-nearest neighbour (k-NN) algorithm. The method was tested on data from eight subjects, achieving an average classification accuracy of 90.25% on held-out, cross-validated data. This demonstrates the efficacy of PSO in optimizing feature selection for cognitive state recognition. The study shows the potential for adaptive workload management based on real-time cognitive monitoring. This approach could significantly enhance performance and reduce cognitive overload.

Cervante et al. [15] proposed two innovative filter feature selection methods that combine Binary Particle Swarm Optimization (BPSO) with information theory for classification tasks. The first method uses BPSO and mutual information to evaluate the relevance and redundancy of feature subsets, while the second method combines BPSO with entropy for a similar purpose. Different weights for relevance and redundancy are applied in the fitness functions to optimize feature selection and classification accuracy. These methods were tested using a decision tree (DT) on four datasets. The results demonstrated that both algorithms, with appropriate weights, could significantly reduce the number of features and achieve similar or better classification accuracy. The first algorithm generally selects a smaller feature subset, whereas the second often results in higher accuracy. This study highlights the effectiveness of integrating BPSO with information theory in enhancing feature selection.

Peng et al. [16] proposed a feature selection method based on the maximal statistical dependency criterion using mutual information, which is challenging to implement directly. They introduced the minimal-redundancy-maximal-relevance (mRMR) criterion as an equivalent form for incremental feature selection. Their two-stage algorithm combines mRMR with advanced feature selectors, such as wrappers, to efficiently select a compact

set of superior features. The method was tested with naive Bayes, support vector machine, and linear discriminant analysis classifiers across four datasets: handwritten digits, arrhythmia, NCI cancer cell lines, and lymphoma tissues. Results showed that mRMR significantly enhanced feature selection and classification accuracy. This approach offers a cost-effective solution for high-quality feature selection in various classification tasks.

Ranjitha et al. [17] addressed the limitations of traditional phishing detection methods, such as URL blacklisting and heuristic-based approaches, which struggle to keep pace with evolving phishing tactics. The study explored the application of machine learning classifiers to identify illegitimate websites, specifically using Multilayer Perceptron and Bernoulli Naive Bayes (NB) classifiers. Feature selection was carried out using a decision tree classifier to identify the most relevant features for adequate classification. The researchers trained and tested their classifiers on a dataset comprising blacklisted and whitelisted websites. Evaluation metrics, including accuracy, precision, recall, and the ROC curve, were used to assess classifier performance. The Multilayer Perceptron achieved an accuracy of over 82%, demonstrating its effectiveness in detecting phishing sites. These findings highlight the potential of machine learning techniques in enhancing phishing detection and mitigating associated risks.

Pan et al. [18] proposed a novel approach to phishing detection that addresses the limitations of existing schemes, which often fail due to the adaptability of phishing attackers. The proposed method identifies anomalies in web pages, specifically discrepancies between a website's identity, structural features, and HTTP transactions. This approach does not require user expertise or prior knowledge of the website, making it more broadly applicable. The research demonstrated that this method poses a high cost to attackers attempting to evade detection. Experimental results showed that the proposed phishing detector achieved a low miss rate and false-positive rate, highlighting its effectiveness in identifying phishing attempts while minimizing errors. This approach provides a promising solution to the ongoing challenge of detecting evolving phishing threats.

Rao et al. [19] addressed the challenge of phishing websites, which often mimic legitimate sites closely, making it difficult for users to distinguish between them. The study focuses on detecting phishing websites by analysing their content and layout. They proposed a novel method that utilizes TF-IDF (Term Frequency-Inverse Document Frequency**)** analysis to extract key terms from suspected phishing sites. These extracted phrases are then queried in various search engines, and the results are integrated and ranked to identify potential phishing sources.

4. Proposed Approach

The proposed method for detecting phishing attacks involves a structured approach beginning with data collection, where relevant data about URLs and their components are gathered from various sources such as databases, websites, and security repositories. This data is then subjected to data preprocessing, which includes cleaning, normalisation, and transformation to ensure consistency and accuracy. Following this, feature selection is conducted using two nature-inspired algorithms, such as Ant Colony Optimization and Dragonfly Algorithm, and two genetic algorithms, namely Particle Swarm Optimization (PSO) and Binary Particle Swarm Optimization (BPSO). Each algorithm attempts to select the most relevant feature subset from the dataset. The selected subset of data is then fitted into clustering models like k-means and Gaussian Mixture Models (GMM). The performance of these clustering methods is evaluated and compared using the silhouette score to determine their effectiveness in accurately identifying phishing attacks.

The development of the machine learning-based predictive model involves the following steps:

2. Data collection
3. Data preprocessing
4. Feature selection
5. Methodology

4.1 Dataset Description

The Phishing Legitimate data set is available both in text and CSV files, which provides the following resources that can be used as inputs for model building: A collection of website URLs for 10000 websites. Each sample has 49 website parameters. These features include URL length, domain age, presence of specific keywords, and structural characteristics. With a binary classification objective, the dataset aims to differentiate between legitimate URLs and those associated with phishing activities. This dataset presents a rich and diverse array of URL attributes, making it suitable for in-depth exploratory analysis and predictive modelling tasks. By examining these features, patterns and insights can be gleaned to enhance cybersecurity measures and mitigate online threats.

Notably, each sample in the dataset includes 49 meticulously documented features, as outlined in Table 4 and Table 5 Description of target class.

Table 4: Description of the dataset.

Attributes	Description
1-15	Features of URL structure
16-22	Features of External Content
22-27	Features of From and Action
28-34	Features of Redirection
34-37	Scripting features
37-42	Features of security
42-47	Domain mismatch features
48-49	URL length metrics

Table 5: Description of the Target class

Dataset	Type
Phishing_legitimate_url	Normal Attack

4.2 Data Pre-processing

Data preprocessing converts raw data into a format suitable for machine learning models. This phase involves using various methods to prepare the dataset for analysis. After the initial preprocessing, the total number of samples decreased to 7,089, and the total number of features reduced to 28. The following sections provide detailed explanations of each preprocessing step for better understanding.

4.2.1 Handling Missing Values

To ensure data quality, an examination was conducted to identify missing values, including NA or blank blocks. To maintain dataset integrity, missing values were to be imputed with the mean of their respective columns. The examination revealed that the values in the dataset are present. As a result, no imputation was necessary. The dataset is complete and ready for the next stage of analysis.

4.2.2 Normalization of URL Length

During the initial exploration of the dataset, it was observed that the feature "URL Length" contained integer values that diverged from the standardised range of -1 to 1, characteristic of other features in the dataset. To maintain uniformity in data scale and facilitate comparative analysis, the "URL Length" was scaled within the desired range of -1 to 1, ensuring consistency across all features.

4.2.3 Identification and Removal of Variance-based Feature

To enhance computational efficiency and simplify the dataset, an evaluation was conducted to identify columns with minimal variance. Columns showing less than 0.01 variance, which indicated a lack of variability, were identified as redundant and removed. This feature reduction helped eliminate unnecessary data, decreasing the dataset's dimensionality. As a result, the dataset was streamlined to include only 32 relevant features, making it more efficient and focused for analysis.

4.2.4 Detection and Removal of Highly Correlated Features

To address multicollinearity and enhance the robustness of the analyses, an examination was conducted to identify pairs of features with high correlation. Features exceeding the correlation threshold of 0.9 were considered highly correlated and thus removed to prevent redundancy. Specifically, the feature "AbnormalExtFormActionR" was identified as correlating higher than 0.9 and dropped from the dataset. This step was crucial in mitigating the risk of overfitting and improving the interpretability of the models trained on the dataset. Removing such redundant information ensures that the model relies on a more independent and diverse set of features.

4.2.5 Outlier Detection and Removal

Outlier detection was conducted on a dataset containing 10,000 samples to enhance the quality and reliability of the data. Statistical methods, including z-score, were employed to identify data points significantly deviating from the expected distribution. These outliers, which could potentially skew the analysis, were systematically removed from the dataset. In total, 2,911 samples were classified as outliers and subsequently eliminated. This process reduced the dataset to 7,089 samples, ensuring a more accurate and representative data set for analysis. The removal of outliers helped prevent these anomalies from unduly influencing the results.

4.2.6 Univariate Feature Analysis and Selection

A thorough univariate analysis was conducted to assess the individual contribution of each feature to the dataset. This analysis aimed to identify features with limited relevance or low informational value. Based on the findings, features deemed less informative were selectively removed. This careful selection process helped streamline the dataset by focusing on the most valuable variables. As a result, 28 relevant features remained after the analysis. Reducing features enhances the dataset's efficiency and focus for subsequent analyses. The remaining features are expected to improve the quality and interpretability of the models. These 28 features were utilized in the next stage of analysis.

4.3 Feature selection

Feature selection is a critical aspect of machine learning as it enhances the accuracy of clustering models. This study compares four feature selection methods the Ant Colony Optimization Algorithm , Dragonfly Algorithm ,PSO and BPSO. After cleaning the data, the dataset is refined to contain 47 features. Subsequently, four different feature selection algorithms are usedto identify and extract the most relevant features from the dataset. Those 32features are implemented in the given algorithm, as shown in Table 6. After the process is completed, the output of the best features by Ant colony is shown in Table 7.

Table 6: Features for Ant Colony Optimization Algorithm

Dataset	**Original Feature**
Phishing url	'NumDots', 'SubdomainLevel', 'PathLevel', 'UrlLength', 'NumDash', 'NumDashInHostname', 'AtSymbol', 'TildeSymbol', 'NumUnderscore', 'NumPercent', 'NumQueryComponents', 'NumAmpersand', 'NumHash', 'NumNumericChars', 'NoHttps', 'RandomString', 'IpAddress', 'DomainInSubdomains', 'DomainInPaths', 'HostnameLength', 'PathLength', 'QueryLength', 'DoubleSlashInPath', 'NumSensitiveWords', 'EmbeddedBrandName', 'PctExtHyperlinks', 'PctExtResourceUrls', 'ExtFavicon', 'InsecureForms', 'RelativeFormAction', 'ExtFormAction', 'AbnormalFormAction',

	'PctNullSelfRedirectHyperlinks', 'FrequentDomainNameMismatch', 'FakeLinkInStatusBar', 'RightClickDisabled', 'PopUpWindow', 'SubmitInfoToEmail', 'IframeOrFrame', 'MissingTitle', 'ImagesOnlyInForm', 'SubdomainLevelRT', 'UrlLengthRT', 'PctExtResourceUrlsRT', 'AbnormalExtFormActionR', 'ExtMetaScriptLinkRT', 'PctExtNullSelfRedirectHyperlinksRT'

Table 7: Features Selection Using Ant Colony Optimization Algorithm

Method	**Selected Features**
Ant Colony Optimization algorithm	'PctNullSelfRedirectHyperlinks', 'TildeSymbol', 'NumNumericChars', 'NumHash', 'NumAmpersand', 'NumQueryComponents', 'NumPercent', 'NumUnderscore', 'AtSymbol', 'RandomString', 'NumDashInHostname', 'NumDash', 'UrlLength', 'PathLevel', 'SubdomainLevel', 'NumDots', 'NoHttps', 'IpAddress', 'AbnormalFormAction', 'EmbeddedBrandName', 'ExtFormAction', 'RelativeFormAction', 'InsecureForms', 'ExtFavicon', 'PctExtResourceUrls', 'PctExtHyperlinks', 'NumSensitiveWords', 'DomainInSubdomains', 'DoubleSlashInPath', 'QueryLength', 'PathLength', 'HostnameLength', 'HttpsInHostname', 'DomainInPaths'

The output of the best features by Dragonfly Algorithm is shown in Table 8.

Table 8: Features Selection Using Dragonfly Algorithm

Method	Selected Features
Dragonfly algorithm	'PctNullSelfRedirectHyperlinks', 'TildeSymbol', 'NumNumericChars', 'NumHash', 'NumAmpersand', 'NumQueryComponents', 'NumPercent', 'NumUnderscore', 'AtSymbol', 'RandomString', 'NumDashInHostname', 'NumDash', 'UrlLength', 'PathLevel', 'SubdomainLevel', 'NumDots', 'NoHttps', 'IpAddress', 'AbnormalFormAction', 'EmbeddedBrandName', 'ExtFormAction', 'RelativeFormAction', 'InsecureForms', 'ExtFavicon', 'PctExtResourceUrls', 'PctExtHyperlinks', 'NumSensitiveWords', 'DomainInSubdomains', 'DoubleSlashInPath', 'QueryLength', 'PathLength', 'HostnameLength', 'HttpsInHostname', 'DomainInPaths'

The output of the best features by PSO is shown in Table 9.

Table 9: Feature Selection Using PSO Algorithm

Method	Selected Features
PSO	'PctNullSelfRedirectHyperlinks', 'TildeSymbol', 'NumNumericChars', 'NumHash', 'NumAmpersand', 'NumQueryComponents', 'NumPercent', 'NumUnderscore', 'AtSymbol', 'RandomString', 'NumDashInHostname', 'NumDash', 'UrlLength', 'PathLevel', 'SubdomainLevel', 'NumDots', 'NoHttps', 'IpAddress', 'AbnormalFormAction', 'EmbeddedBrandName', 'ExtFormAction', 'RelativeFormAction', 'InsecureForms', 'ExtFavicon', 'PctExtResourceUrls', 'PctExtHyperlinks', 'NumSensitiveWords', 'DomainInSubdomains', 'DoubleSlashInPath', 'QueryLength', 'PathLength', 'HostnameLength', 'HttpsInHostname', 'DomainInPaths'

The output of the best features by BPSO is shown in Table 10.

Table 10: Features Selection Using BPSO Algorithm

Method	Selected Features
BPSO	'RandomString', 'NumDashInHostname', 'NumDash', 'UrlLength', 'PathLevel', 'SubdomainLevel', 'NumDots', 'NoHttps', 'IpAddress', 'AbnormalFormAction', 'EmbeddedBrandName', 'ExtFormAction', 'RelativeFormAction', 'InsecureForms', 'ExtFavicon', 'PctExtResourceUrls', 'PctExtHyperlinks', 'NumSensitiveWords', 'DomainInSubdomains', 'DoubleSlashInPath', 'QueryLength', 'PathLength', 'HostnameLength', 'HttpsInHostname', 'DomainInPaths"PctNullSelfRedirectHyperlinks', 'TildeSymbol', 'NumNumericChars', 'NumHash', 'NumAmpersand', 'NumQueryComponents', 'NumPercent', 'NumUnderscore', 'AtSymbol'

The output of the best features by Univariate Analysis is shown in Table 10.

Table 11: Features Selection Using Univariate Analysis

Method	**Selected Features**
Univariate Analysis	'RandomString', 'NumDashInHostname', 'NumDash', 'UrlLength', 'PathLevel', 'SubdomainLevel', 'NumDots', 'NoHttps', 'IpAddress', 'AbnormalFormAction', 'EmbeddedBrandName', 'ExtFormAction', 'RelativeFormAction', 'InsecureForms', 'ExtFavicon', 'PctExtResourceUrls', 'PctExtHyperlinks', 'NumSensitiveWords', 'DomainInSubdomains', 'DoubleSlashInPath', 'QueryLength', 'PathLength', 'HostnameLength', 'HttpsInHostname', 'DomainInPaths"PctNullSelfRedirectHyperlinks', 'TildeSymbol', 'NumNumericChars', 'NumHash', 'NumAmpersand', 'NumQueryComponents', 'NumPercent', 'NumUnderscore', 'AtSymbol'

4.4. Methodology

The framework of the clustering model is presented in Figure 5.

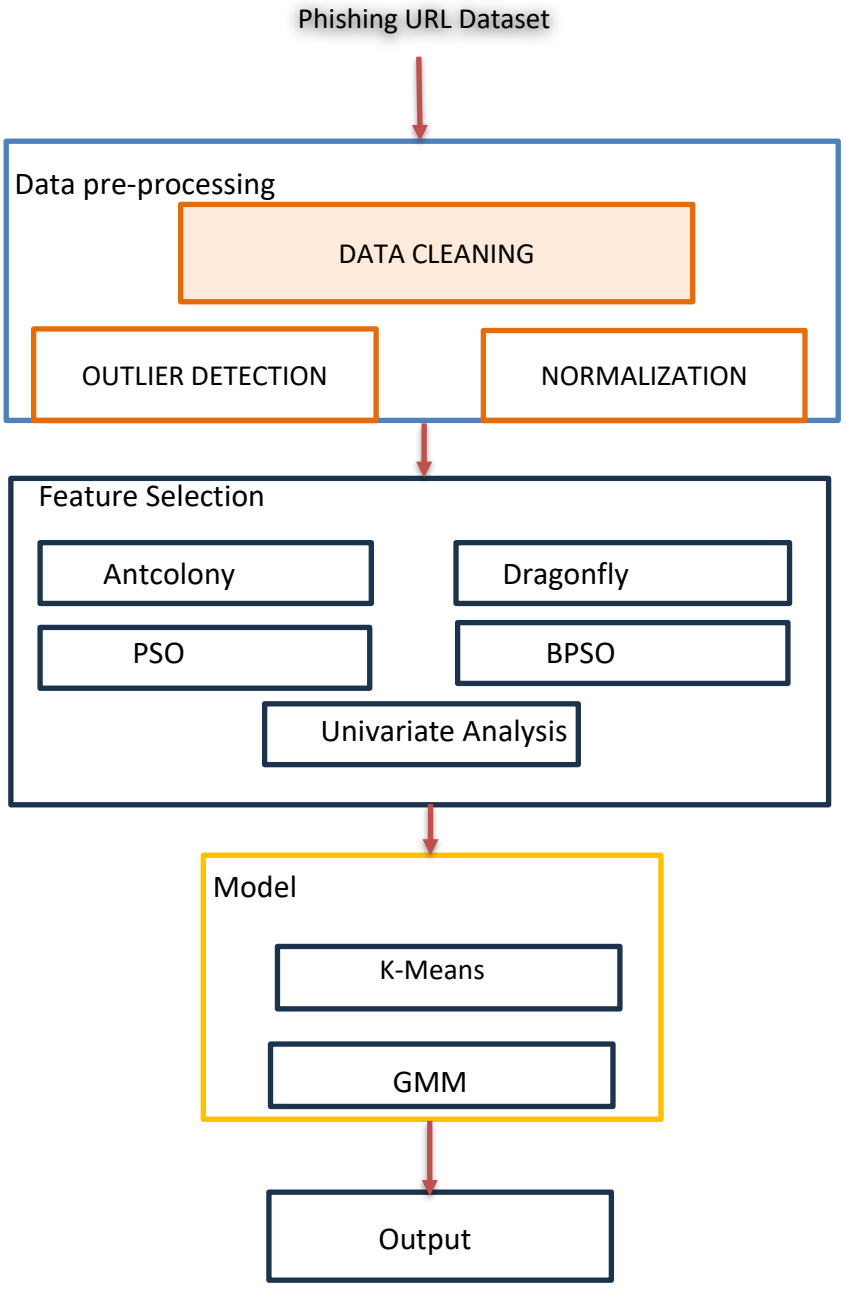

Figure 5: Framework of the Clustering Model.

Optimizers and Model

Four optimizers are employed in the feature selection: two nature-inspired algorithms, ant colony optimization and dragonfly algorithm, and two genetic algorithms, PSO and BPSO. Each optimizer has a range of hyperparameters that significantly influence its performance.

Using nature-inspired and genetic algorithms for feature selection is effective because they explore large and complex search spaces, avoiding local optima. These algorithms focus on global optimization, increasing the likelihood of finding the best feature subset. Their flexibility allows them to adapt to different data types and optimization challenges. Additionally, they are robust to noise and irrelevant features, improving model accuracy and generalization. The ant colony optimization, inspired by ants' foraging behaviour, involves parameters like the number of ants, pheromone evaporation rate, and pheromone influence. The dragonfly algorithm, modelled after dragonflies' static and dynamic swarming behaviours, includes hyperparameters such as alignment, cohesion, separation weights, and attraction to food sources. On the genetic algorithm side, PSO mimics the social behaviour of birds flocking or fish schooling, using parameters like cognitive and social coefficients, inertia weight, and the number of particles. BPSO, a variant tailored for binary search spaces, uses similar parameters but adapts them to binary decisions, impacting the velocity and position updates in a discrete space. To tune these hyperparameters effectively, an exploratory approach is employed. This involves systematically experimenting with different parameter values to uncover the optimal configurations that yield the best performance. The goal is to identify the most relevant subset of hyperparameters that enhance the efficiency and accuracy of each optimizer. Once the optimal hyperparameters are determined, the dataset is preprocessed and split into training and testing sets. The preprocessed and feature-selected dataset is then fitted into unsupervised learning models like k-means and GMM. K-means clustering relies on the distance between points to form clusters, while GMM assumes that the data is generated from a mixture of several Gaussian distributions. Using GMM for clustering is advantageous because it models data as a mixture of Gaussian distributions, allowing for more flexible, elliptical cluster shapes. GMM captures complex underlying structures in the data significantly when clusters overlap. The Expectation-Maximization (EM) algorithm used by GMM iteratively improves the fit to the data, enhancing clustering accuracy.

6. Results and Analysis

Python 3.10 is considered for implementation in this study due to its wide range of libraries. Various machine learning algorithms are employed to categories the binary-class network attacks. The Phishing URL attack dataset is obtained from the KAGGLE repository for model implementation. The dataset is pre-processed, and relevant features are selected from the dataset. The dataset is then split into training-validation and testing sets in a 75%—25% ratio for model training and testing. Unsupervised machine learning models, namely, K-means and GMM, are built using PyTorch. The following listed hyperparameter values of PSO are used to select the most relevant features from the dataset.

C1 =1: A value of 1 indicates a moderate influence, meaning the particle will consider its own past experiences but not be overly reliant on them.

C2 = 1.5: A value of 1.5 suggests a more substantial influence from the global best compared to the personal best, encouraging particles to follow the collective wisdom of the swarm more than their own experience.

w = .9: An inertia weight of 0.9 implies that the particles will maintain most of their momentum from the previous iteration, resulting in smoother and more gradual updates in their velocity. This can help in exploring the search space more effectively.

v= 1 Setting the initial velocity v to 1 means that each particle in the swarm starts with a velocity of 1.

K=20: A value of 20 provides a good balance between exploration and exploitation.

Table 12 presents listed hyperparameter values of PSO are used to select the most relevant features from the dataset.

Table 12: Hyperparameters value based on exploration optimization of PSO.

Hyperparameters	**Values**
c1	1
c2	1.5
w	.9
v	1
k	20

The following listed hyperparameter values of Binary PSO are used to select the most relevant features from the dataset.

C1 =1.5 A value of 1.5 indicates a moderate influence, meaning the particle will consider its own past experiences but not be overly reliant on them.

C2 = 1.5: A value of 1.5 suggests a more substantial influence from the global best compared to the personal best, encouraging particles to follow the collective wisdom of the swarm more than their own experience.

w =0.7: An inertia weight of 0.7 implies that the particles will maintain most of their momentum from the previous iteration, resulting in smoother and more gradual updates in their velocity. This can help in exploring the search space more effectively.

Effect on Convergence: Lower values for r1 and r2 make the algorithm more exploitative, relying more on the best-known positions. Higher values increase exploration, allowing particles to search more widely. These specific hyperparameter values are chosen based on exploring a dataset such as PSO or binary PSO optimization. During this process, different combinations ofparameter values are tested, and the combination that results in the best model performance on a validation set is selected.

The hyperparameter values listed below of ant colony are used to optimize the selection of features from the dataset.

Table 13 presents listed hyperparameter values of BPSO are used to select the most relevant features from the dataset.

Table 13: Hyperparameters value based on exploration optimization of Binary PSO.

Hyperparameters	**value**
C1	1.5
C2	1.5
W	.7
r 1	0.854
r 2	0.75

The following hyperparameter values of Antcolony Algorithm are used to select the most relevant features from the dataset.

Number of Ants (n_ants = 10): In each iteration, 10 ants explore the solution space.

Number of Iterations (n_iterations = 50): The algorithm runs for 50 iterations.

Pheromone Importance (alpha = 1): Pheromone trails moderately influence path selection.

Heuristic Importance (beta = 2): Heuristic information (e.g., path length) is highly influential in decision-making.

Evaporation Rate (roh): 50% of the pheromone evaporates each iteration, balancing exploration and exploitation.

These specific hyperparameter values are chosen based on exploring a Dataset in this ant colony optimisation. During this process, different combinations ofparameter values are tested, and the combination that results in the best model performance on a validation set is selected. The hyperparameter values of the dragonfly algorithm, listed below optimize the selection of features from the dataset.

Table 14 presents listed hyperparameter values of Antcolony are used to select the most relevant features from the dataset.

Table 14: Hyperparameters value based on exploration optimization of Ant Colony Algorithm.

Hyperparameters	**Values**
n	10
N	50
alpha	1
beta	2
roh	50

The following hyperparameter values of Dragonfly Algorithm are used to select the most relevant features from the dataset.

Num_featurs (N): 28 features define the dimensions of the problem space.

Max_generations(m): 1000: The algorithm is executed for 1000 generations. Setting the maximum number of generations to 1000 allows the algorithm sufficient time to explore and refine potential solutions. This value balances exploration and convergence, providing ample opportunity to find an optimal solution without excessively prolonging the computation.

Population_size (p): 250: A population size of 250 provides a good balance between exploration and computational efficiency. It is large enough to explore the solution space effectively.

Step_size (s): 0.2: A step size of 0.2 allows for moderate adjustments to the dragonflies' positions. It helps them explore the search space without making overly large jumps.

absorption_coffeicient (c):.6: An absorption coefficient of 0.6 indicates a moderate level of absorption. This value strikes a balance between adapting to the environment and relying on the best global information. It helps maintain a balance between exploration and exploitation.

These specific hyperparameter values are chosen based on exploring a dataset in this dragonfly algorithm optimization. During this process, different combinations of parameter values are tested, and the combination results in the best model performance on a validation set.

Table 15 presents listed hyperparameter values of Dragonfly Algorithm are used to select the most relevant features from the dataset.

Table 15: Hyperparameters value based on exploration optimization of Dragonfly Algorithm

Hyperparameters	Values
N	28
m	1000
p	250
s	.2
c	.6

The comparative analysis of the silhouette scores achieved by the k-means and GMM models for feature sets selected using Ant Colony, Dragonfly, PSO, BPSO optimizers, and univariate analysis is provided in Table 1. The BPSO-selected feature subset yielded the highest silhouette score when used with the k-means model.

Table 16: Silhouette scores of Clustering Algorithms based on Four Optimizers and Univariate Analysis.

Algorithms / **Models**	**Ant Colony**	**PSO**	**BPSO**	**Dragonfly**	**Univariate Analysis**
K-means	76%	76%	87%	65%	80%
GMM	57%	59%	58%	55%	65%

6. Conclusion and Future Scope

This study has demonstrated the potential of unsupervised machine-learning techniques in detecting phishing URLs. Focusing on patterns and anomalies within URL data shows that K-means clustering and Gaussian Mixture Models can effectively identify suspicious URLs without needing labelled data. The findings suggest that these unsupervised algorithms can overcome some limitations of traditional supervised methods and offer a viable alternative for enhancing cybersecurity measures. This study highlights the importance of leveraging the inherent structure of URLs to detect phishing attempts, contributing to the broader effort to protect digital assets. Future work could explore combining unsupervised and supervised learning to improve detection accuracy. Enhancing feature extraction techniques, possibly using deep learning, could make models more robust. Investigating real-time data integration and adaptive learning could help systems respond quickly to new threats. Additionally, expanding this work to include multiple data sources and cross-domain analysis could increase the applicability and effectiveness of these techniques in various cybersecurity contexts. Lastly, developing user-friendly tools based on these findings could make advanced phishing detection accessible to a broader range of users and organizations.

References

[1]. M. Khonji, Y. Iraqi and A. Jones, “Phishing detection: a literature survey”, IEEE Communication, pp.2091–2121, 2013.

[2]. R. S. Rao, A. Pais, “Detection of phishing websites using an efficient feature-based machine learning framework”, Neural Computing and Applications, pp. 3851–3873, 2019.

[3]. L. Yu and H. Liu, “Efficient feature selection via analysis of relevance and redundancy,” Journal of Machine Learning Research, vol. 5, pp. 1205–1224, Jan. 2004.

[4]. R. S. Parpinelli, H. S. Lopes and A. A. Freitas, “Data mining with an ant colony optimization algorithm”, IEEE Transactions on Evolutionary Computation, pp.321-332,2002.

[5]. Y. Meraihi, A. Ramdane-Cherif and D. Acheli, “Dragonfly algorithm: a comprehensive review and applications”, Neural Computing and Applications**,** vol.32, pp.16625–16646 ,2020.

[6]. J. Kennedy, and R. C. Eberhart, “Particle swarm optimization”, IEEE International Conference on Neural Networks, Vol. 4, pp. 1942-1948,1995.

[7]. M. A. Khanesar, M Teshnehlab and M. A. Shoorehdeli, "A novel binary particle swarm optimization”, Mediterranean Conference on Control & Automation, pp. 1-6, 2007.

[8]. K. P. Sinaga and M. -S. Yang, "Unsupervised K-Means Clustering Algorithm”, IEEE Access, vol. 8, pp. 80716-80727, 2020.

[9]. H. Wan, H. Wang, B. Scotney and J. Liu, "A Novel Gaussian Mixture Model for Classification," 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), Bari, Italy, 2019, pp. 3298-3303.

[10]. A. C. Bahnsen, E. C. Bohorquez “Classifying phishing URLs using recurrent neural networks”, APWG Symposium on Electronic Crime Research (eCrime), IEEE, pp. 1–8, 2017.

[11]. J. Li et al., “Feature selection: A data perspective,” ACM Computing Surveys (CSUR), vol. 50, no. 6, pp. 1–45, 2017.

[12]. A. Al-Ani, “Feature Subset Selection Using Ant Colony Optimization”, International Journal of Computational Intelligence, pp 53-58, 2005.

[13]. D. V. Sekhar, M. P. Reddy, “Feature Selection Based on Dragonfly Optimization for Psoriasis Classification”, International Journal of Intelligent Systems and Applications in Engineering, pp. 935–943, 2024.

[14]. H. A. Firpi and R. J. Vogelstein, "Particle swarm optimization-based feature selection for cognitive state detection”, Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Boston, MA, USA, 2011, pp. 6556-6559,2011.

[15]. L. Cervante, Bing Xue, M. Zhang and Lin Shang, "Binary particle swarm optimization for feature selection: A filter-based approach," IEEE Congress on Evolutionary Computation, Brisbane, pp. 1-8,2012.

[16]. H. Peng, F. Long, and C. Ding, “Feature selection based on mutual information criteria

of max-dependency, max-relevance, and min redundancy," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pp. 1226–1238, Aug. 2005.

[17]. R. G. Ranjitha and M. Sundaram, "Phishing Websites Classification Placed on URL Features and Extreme Machine Learning", Federated Engineering and Systems, PP.37–43, 2023.

[18]. Y. Pan, X. Ding, "Anomaly-based web phishing page detection", 22nd Annual Computer Security Applications Conference (ACSAC'06), IEEE, pp. 381–392, 2006.

[19]. R. S. Rao, A. Pais "An enhanced blacklist method to detect phishing websites", International Conference on Information Systems Security. Springer, pp. 323–333,2017.

[20]. D. Terence, " An optimality principle for unsupervised learning", NIPS, pp. 11–19,1988.

[21]. M. Dorigo, T. Sttzle, and M. Birattari, "Ant colony optimization", Computational Intelligence Magazine, 1(4), pp.28-39,2006.

[22]. B. Xue, M. Zhang, and W. N. Browne, "Particle swarm optimization for feature selection in classification: A multi-objective approach," IEEE Transactions on Cybernetics, vol. 43, no. 6, pp. 1656–1671, Dec. 2013.

Appendix

Code

ANTCOLONY

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import networkx as nx

class AntColonyFeatureSelectionUnsupervised:
    def __init__(self, graph, n_ants, n_iterations, alpha=1, beta=2, evaporation_rate=0.1):
        self.graph = graph
        self.n_ants = n_ants
        self.n_iterations = n_iterations
        self.alpha = alpha
        self.beta = beta
        self.evaporation_rate = evaporation_rate
        self.pheromone_trails = np.ones((graph.number_of_nodes(), graph.number_of_nodes()))
        self.n_features = graph.number_of_nodes()
        self.pheromone_delta = np.zeros(self.n_features)

    def fit(self, X_train):
        self.n_features = X_train.shape[1]
        self.pheromone_matrix = np.ones(self.n_features) * 0.5

        for _ in range(self.n_iterations):
            selected_features = []
            for _ in range(self.n_ants):
                features = self.construct_solution()
                selected_features.append(features)
            self.update_pheromones(selected_features, X_train)

    def construct_solution(self):
        features = []
        remaining_features = list(range(self.n_features))
        while remaining_features:
            probabilities = self.calculate_probabilities(remaining_features)
            selected_feature = np.random.choice(remaining_features, p=probabilities)
            features.append(selected_feature)
            remaining_features.remove(selected_feature)
        return features

    def calculate_probabilities(self, remaining_features):
        probabilities = [self.pheromone_matrix[feature] for feature in remaining_features]
        total_pheromone = sum(probabilities)
        return [pheromone / total_pheromone for pheromone in probabilities]

    def update_pheromones(self, selected_features_list, X_train):
        pheromone_delta = np.zeros(self.n_features)
        for selected_features in selected_features_list:
            X_selected = X_train.iloc[:, selected_features]
            kmeans = KMeans(n_clusters=2, random_state=0,n_init=10).fit(X_selected)
            silhouette_avg = silhouette_score(X_selected, kmeans.labels_)
            for feature in selected_features:
                pheromone_delta[feature] += silhouette_avg
        self.pheromone_matrix = (1 - self.evaporation_rate) * self.pheromone_matrix + pheromone_delta

def create_graph(X_train):
    corr_matrix = np.abs(np.corrcoef(selected_df, rowvar=False))
    graph = nx.Graph()
    for i in range(selected_df.shape[1]):
        for j in range(i+1, selected_df.shape[1]):
            graph.add_edge(i, j, weight=corr_matrix[i, j])
    return graph
```

```
# Initialize and fit the ACO feature selection algorithm
graph = create_graph(selected_df)
aco_unsupervised = AntColonyFeatureSelectionUnsupervised(graph=graph, n_ants=10, n_iterations=50)
aco_unsupervised.fit(selected_df)

# Select the best features based on pheromone trails
pheromone_matrix_array = np.asarray(aco_unsupervised.pheromone_matrix)
best_features_indices = np.argsort(pheromone_matrix_array)[::-1]  # Select top 10 features
best_features_indices
print("Selected features indices:", best_features_indices)
```

Selected features indices: [34 8 14 13 12 11 10 9 7 16 6 5 4 3 2 1 15 17 33 26 32 31 30 29 28 27 25 18 24 23 22 21 20 19]

['PctNullSelfRedirectHyperlinks', 'TildeSymbol', 'NumNumericChars', 'NumHash', 'NumAmpersand', 'NumQueryComponents', 'NumPercent', 'NumUnderscore', 'AtSymbol', 'RandomString', 'NumDashInHostname', 'NumDash', 'UrlLength', 'PathLevel', 'SubdomainLevel', 'NumDots', 'NoHttps', 'IpAddress', 'AbnormalFormAction', 'EmbeddedBrandName', 'ExtFormAction', 'RelativeFormAction', 'InsecureForms', 'ExtFavicon', 'PctExtResourceUrls', 'PctExtHyperlinks', 'NumSensitiveWords', 'DomainInSubdomains', 'DoubleSlashInPath', 'QueryLength', 'PathLength', 'HostnameLength', 'HttpsInHostname', 'DomainInPaths']

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import silhouette_score
best_features=selected_df.iloc[:, best_features_indices]
kmeans=KMeans(n_clusters=2,init="k-means++")
y_labels=kmeans.fit_predict(best_features)
silhouette_avg = silhouette_score(best_features, y_labels)
print(f'Silhouette Score: {silhouette_avg}')
#accuracy = accuracy_score(y_train, y_labels)  # Adjust for the clustering labels
#print(f"Accuracy Score: {accuracy}")
```

Silhouette Score: 0.7604676567184461

#Dragonfly

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.cluster import MiniBatchKMeans


class UnsupervisedDragonflyAlgorithm:
    def __init__(self, num_features, max_generations=1100, population_size=250,
step_size=0.2, absorption_coefficient=0.6):
        self.num_features = num_features
        self.max_generations = max_generations
        self.population_size = population_size
        self.step_size = step_size
        self.absorption_coefficient = absorption_coefficient
        self.population = None
        self.best_individual = None

    def initialize_population(self):
        self.population = np.random.rand(self.population_size, self.num_features) > 0.6

    def evaluate_fitness(self, X):
        fitness_values = []
        for individual in self.population:
            selected_features = [i for i in range(self.num_features) if individual[i] == 1]
            X_selected = selected_df.iloc[:, selected_features]

            kmeans = KMeans(n_clusters=2, random_state=0).fit(X_selected)
            silhouette_avg = silhouette_score(X_selected, kmeans.labels_)
            fitness_values.append(silhouette_avg)
        return fitness_values
```

```
    def evolve(self, X):
        self.initialize_population()
        for generation in range(self.max_generations):
            fitness_values = self.evaluate_fitness(X)
            best_index = np.argmax(fitness_values)
            self.best_individual = self.population[best_index]
            new_population = []
            for individual in self.population:
                new_individual = individual.astype(bool) + self.step_size *
(self.best_individual != individual) + \
                                 self.absorption_coefficient *
np.random.randn(self.num_features)
                new_population.append(new_individual)
            self.population = np.clip(new_population, 0, 1)


# Feature selection using Unsupervised Dragonfly Algorithm
uda = UnsupervisedDragonflyAlgorithm(num_features=selected_df.shape[1], population_size=10,
max_generations=50)
uda.evolve(selected_df)

# Get the selected features from the best individual
selected_features = [i for i in range(selected_df.shape[1]) if uda.best_individual[i] == 1]
print("Selected features:", selected_features)
```

Selected features: [1, 2, 3, 5, 6, 7, 8, 11, 12, 14, 16, 18, 20, 22, 27, 29, 30, 31, 32, 33]

```
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import silhouette_score
#best_features=X1.columns[best_features_indices ]
best_features_indices = np.array( selected_features).reshape(-1, 1)
#best_features_indices = selected_features.reshape(-1, 1)
kmeans=KMeans(n_clusters=2,init="k-means++")
y_labels=kmeans.fit_predict(best_features_indices)
silhouette_avg = silhouette_score(best_features_indices, y_labels)
print(f'Silhouette Score: {silhouette_avg}')
```

Silhouette Score: 0.6509154018350569

#PSO

```
import numpy as np
import pandas as pd
import pyswarms as ps
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import silhouette_score
def objective_function(selected_features):
    selected_indices = np.where(selected_features == 1)[0]
    if len(selected_indices) == 0:
        return -np.inf



    # Train a classifier
    # Use K-means clustering
    kmeans = KMeans(n_clusters=2, random_state=0)
    cluster_labels = kmeans.fit_predict(selected_df.iloc[selected_indices])

    # Calculate Silhouette Score
    silhouette = silhouette_score(selected_df.iloc[selected_indices], cluster_labels)

    return silhouette

# Define PSO parameters
num_features = selected_df.shape[1]
num_particles = 20
num_iterations = 150
options = {'c1':1 , 'c2': 1.5, 'w':.9, 'k': 20, 'p': 1, 'bounds': (np.zeros(num_features),
np.ones(num_features))}

bounds = (np.zeros(num_features), np.ones(num_features))

# Initialize PSO optimizer
optimizer = ps.discrete.binary.BinaryPSO(n_particles=num_particles, dimensions=num_features,
options=options,)

best_position, _ = optimizer.optimize(objective_function, iters=num_iterations)

selected_indices = np.where(np.atleast_1d(_ == 1))[0]
#selected_indices = np.where(np.atleast_1d(_) > 0.20)[0]

selected_indices = np.clip(selected_indices, 0, selected_df.shape[1] - 1)
print("best_position",best_position )
print("best",selected_indices)

top_30_indices = selected_indices#[:47]
top_30_features = selected_df.columns[top_30_indices]

# Print the selected features
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
best_features=selected_df.iloc[:,top_30_indices ]
kmeans=KMeans(n_clusters=2,init="k-means++")
y_labels=kmeans.fit_predict(best_features)
silhouette_avg = silhouette_score(best_features, y_labels)
print(f'Silhouette Score: {silhouette_avg}')
```

Selected features: [4 5 6 7 9 12 13 14 24 25 29 32 33 34]
Silhouette Score: 0.767803452459353

#Binary PSO

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

class Particle:
    def __init__(self, num_features):
        self.position = np.random.randint(2, size=num_features)
        self.velocity = np.random.uniform(-0.1, 0.1, size=num_features)
        self.best_position = self.position.copy()
        self.best_score = float('-inf')  # Initialize with negative infinity for
maximization

def fitness_function(features, X):
    selected_features = [bool(f) for f in features]
    X_selected = selected_df.iloc[:, selected_features]

    kmeans = KMeans(n_clusters=2, random_state=0,n_init=10)
    kmeans.fit(X_selected)
    silhouette = silhouette_score(X_selected, kmeans.labels_)

    return silhouette

def bell_pso(X, num_particles=10, max_iter=50, w=0.7, c1=1.5, c2=1.5):
    num_features = X.shape[1]
    particles = [Particle(num_features) for _ in range(num_particles)]
    global_best_position = np.zeros(num_features)
    global_best_score = float('-inf')  # Initialize with negative infinity for maximization

    for _ in range(max_iter):
        for particle in particles:
            fitness = fitness_function(particle.position, X)
            if fitness > particle.best_score:
                particle.best_position = particle.position.copy()
                particle.best_score = fitness

            if fitness > global_best_score:
                global_best_position = particle.position.copy()
                global_best_score = fitness

        for particle in particles:
            r1 = np.random.rand(num_features)
            r2 = np.random.rand(num_features)
            particle.velocity = w * particle.velocity + c1 * r1 * (particle.best_position -
particle.position) + c2 * r2 * (global_best_position - particle.position)
            particle.position = np.round(1 / (1 + np.exp(-particle.velocity)))

    return global_best_position

# Example usage
# Assuming X1 is your dataset
#X1 = np.random.rand(100, 10)  # Example random dataset

selected_features = bell_pso(selected_df, num_particles=10, max_iter=50)
print("Selected features:", selected_features)
selected_features_indices = np.where(selected_features == 1)[0]
print("Selected features indices:", selected_features_indices)
```

Selected features indices: [6 11 14 17 18 19 22 25 27 28 29 30 34]

Silhouette Score(k-means): 0.87