Natural-inspired Pattern Recognition for Classification Problem: Hyperparameter Optimization (HPO) of Machine Learning Models - Traditional Algorithms versus Natural-inspired Algorithms

Mohammed

Abdelfattaahh

May 04, 2025

Project Idea in Detail

This project investigates the application of natural-inspired pattern recognition techniques for addressing classification problems, with a specific focus on hyperparameter optimization (HPO) of machine learning models. The primary objective is to compare traditional optimization algorithms, such as grid search and random search, with nature-inspired algorithms, including genetic algorithms, particle swarm optimization, and ant colony optimization. the project aims to enhance the performance of an SVM classifier on the MNIST dataset by optimizing its hyperparameters using a variety of HPO techniques. The MNIST dataset, a benchmark for handwritten digit recognition, provides a standardized platform to evaluate the effectiveness of different optimization strategies. The HPO methods under investigation include two traditional approaches—GridSearchCV and Random Search—and a nature-inspired algorithm, Bacterial Foraging Optimization (BFO), along with its variations: Standard BFO, Dynamic BFO, BFO with local search, Binary BFO, and BFO with dynamic step size and mutation. The primary objective is to determine which HPO method achieves the highest classification accuracy while maintaining computational efficiency, as measured by training and optimization time

The motivation for this project stems from the critical role of hyperparameters in determining the performance of SVM models. Hyperparameters such as the regularization parameter C C and the kernel coefficient γ γ (for the RBF kernel) significantly influence the model's ability to generalize to unseen data. Traditional HPO methods like GridSearchCV exhaustively search the hyperparameter space, which can be computationally expensive, while Random Search samples randomly, potentially missing optimal configurations. BFO, inspired by the foraging behavior of bacteria, offers a heuristic approach that may navigate the hyperparameter space more efficiently. By including variations of BFO, the project seeks to explore whether modifications to the standard algorithm can further enhance its performance, providing insights into the applicability of nature-inspired methods in machine learning optimization

Main Functionalities

The project encompasses the following key functionalities:

- **Hyperparameter Optimization using Evolutionary Algorithms:**
- This functionality involves implementing and applying multiple HPO methods to tune the SVM's hyperparameters. GridSearchCV systematically evaluates all combinations within a predefined grid of hyperparameter values, while Random Search samples a fixed number of configurations randomly. BFO and its variations (Standard BFO, Dynamic BFO, BFO with local search, Binary BFO, and BFO with dynamic step size and mutation) are used to search for optimal hyperparameters by mimicking bacterial foraging behavior, adapting the search process through chemotaxis, reproduction, and elimination-dispersal steps. Each BFO variation introduces modifications to improve convergence or exploration, such as adaptive step sizes or local search mechanisms.

- **Model Training:** The SVM model is trained on the MNIST dataset using both default hyperparameters (as provided by scikit-learn) and the optimized hyperparameters obtained from each HPO method. The training process involves fitting the SVM classifier to the 60,000 training images and evaluating its performance on the 10,000 test images. The use of a single model type (SVM) ensures a controlled comparison across different HPO methods.

- **Performance Evaluation:** The performance of the SVM model under each hyperparameter configuration is assessed using two metrics: classification accuracy on the test set and computational time, which includes both the time required for HPO and model training. Accuracy measures the proportion of correctly classified digits, while computational time reflects the efficiency of the optimization process. This dual evaluation provides a comprehensive view of each method's effectiveness, balancing predictive performance with practical considerations.

These functionalities form a streamlined pipeline that focuses on the core aspects of HPO and model evaluation, aligning with the user's request to exclude preprocessing and feature extraction.

Similar Applications in the Market

Several tools and frameworks in the market address hyperparameter optimization and automated machine learning, providing a context for this project:

- **Optuna:** An open-source framework that allows users to define an objective function for optimization and employs techniques like Bayesian optimization to find optimal hyperparameters (https://optuna.org/).
- **Hyperopt:** A Python library designed for optimizing complex search spaces, supporting real-valued, discrete, and conditional hyperparameters (http://hyperopt.github.io/hyperopt/).
- **Auto-sklearn:** An automated machine learning toolkit that integrates preprocessing pipelines with machine learning algorithms and optimizes them using Bayesian optimization (https://automl.githu sklearn/master/).

While these tools predominantly rely on traditional optimization methods, there is increasing interest in incorporating nature-inspired algorithms for HPO due to their potential to explore search spaces more efficiently. This project aligns with this trend by focusing on evolutionary algorithms for HPO in classification tasks.

however, the use of nature-inspired algorithms like BFO is less common in commercial tools, with most applications appearing in academic research. Studies have demonstrated the potential of BFO for optimizing parameters in machine learning models, These research efforts suggest that BFO can offer advantages in exploring complex search spaces, making this project's focus on BFO and its variations a novel contribution to the field. By comparing BFO with standard tools like GridSearchCV and Random Search, the project bridges the gap between traditional and nature-inspired approaches, potentially highlighting new opportunities for HPO in practical applications.

Literature Review of Academic Publications

To provide a robust theoretical foundation, the project draws upon several academic publications in the fields of hyperparameter optimization, bio-inspired computing, and pattern recognition:

An enhanced Bacterial Foraging Optimization and its application : This study proposes an improved BFO algorithm with chaotic chemotaxis, Gaussian mutation, and local search for optimizing parameters in extreme learning machines. It showcases BFO's applicability to machine learning optimization tasks, supporting its use in this project.

Improved bacterial foraging optimization with deep learning based anomaly detection : This paper applies an improved BFO for feature selection and hyperparameter optimization in deep learning models, illustrating BFO's versatility and effectiveness in complex optimization problems.

Parameter Optimization of Extreme Learning Machine Using Bacterial Foraging Algorithm (http://koreascience.or.kr/article/JAKO200714364664411.page):This study uses BFO to optimize the input weights and biases of an Extreme Learning Machine, providing a direct precedent for applying BFO to machine learning parameter optimization.

- **Genetic Algorithms for Hyperparameter Optimization:** Research demonstrates that genetic algorithms are effective for optimizing hyperparameters in machine learning models. For example, the study "Efficient Hyperparameter Optimization in Deep Learning Using a Variable Length Genetic Algorithm" explores the use of a variable-length genetic algorithm to tune hyperparameters of convolutional neural networks, achieving improved performance compared to traditional methods [1].
- **Bio-inspired Computing in Classification:** Bio-inspired models, such as spiking neural P systems, have been applied to classification problems. The paper "A learning numerical spiking neural P system for classification problems" introduces a novel approach that leverages biological neural network principles for pattern recognition tasks, reporting enhanced recognition accuracy [2].

- **Pattern Recognition with Nature-inspired Methods:** Literature highlights the potential of nature-inspired algorithms, such as swarm intelligence and evolutionary computation, in enhancing pattern recognition. These methods have been used to develop robust feature extraction and selection techniques, which are critical for effective classification.

These studies collectively underscore the potential of nature-inspired algorithms to address challenges in classification and hyperparameter optimization, providing a strong rationale for the project's approach.

Dataset Employed

The project utilizes a publicly available dataset to evaluate the performance of the proposed natural-inspired HPO methods. A suitable choice is the MNIST dataset, a widely recognized benchmark for classification tasks. The MNIST dataset comprises 70,000 handwritten digit images, each representing a digit from 0 to 9. It is structured as follows:

Table 1: MNIST Dataset Structure

Attribute	Description
Training Set	60,000 images
Test Set	10,000 images
Image Size	28x28 pixels
Classes	10 (digits 0 through 9)

The simplicity and well-defined nature of the MNIST dataset make it an ideal choice for comparing different HPO strategies, as it allows for clear evaluation of the optimization methods' effectiveness without the complexities introduced by more intricate datasets.

The MNIST dataset's simplicity and standardized format make it an ideal choice for evaluating HPO methods, as it allows for clear comparisons without the complexities of more intricate datasets. Its public availability through libraries like scikit-learn ensures reproducibility, and its widespread use in machine learning research enables benchmarking against existing studies.

Details of the Algorithm(s)/Approach(es) Used and the Results of the Experiments :

The experimental approach involves a systematic comparison of HPO methods for tuning an SVM classifier on the MNIST dataset. The methodology is outlined as follows :

Initial Training with Default Hyperparameters: An SVM classifier, implemented using scikit-learn's SVC class, is trained on the MNIST dataset with its default hyperparameters(C , kernel type and gamma). The classification accuracy on the test set and the training time are recorded to establish a baseline.

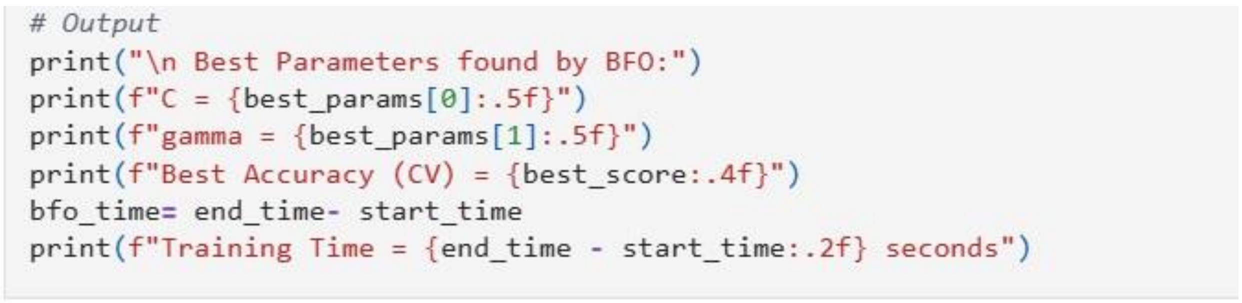
Hyperparameter Optimization:
GridSearchCV: A grid of hyperparameter values for C (e.g., [0.1, 1, 10, 100]) and gammaγ (e.g., [0.001, 0.01, 0.1, 1]) is defined, and GridSearchCV evaluates all combinations using cross-validation to select the best configuration.

Random Search: A fixed number of hyperparameter combinations are sampled randomly from the same search space as GridSearchCV, offering a more efficient alternative.

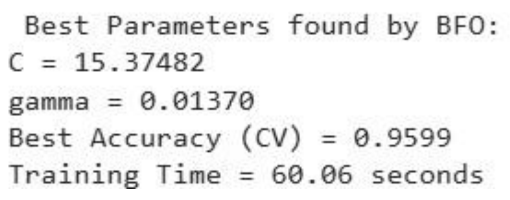
BFO and Variations:

BFO is adapted to optimize C, gamma, and possibly the kernel type by treating them as variables in a continuous or discrete search space. The variations include:

Standard BFO Results: best parameters found and Training Time



```
# Output
print("\n Best Parameters found by BFO:")
print(f"C = {best_params[0]:.5f}")
print(f"gamma = {best_params[1]:.5f}")
print(f"Best Accuracy (CV) = {best_score:.4f}")
bfo_time= end_time- start_time
print(f"Training Time = {end_time - start_time:.2f} seconds")
```

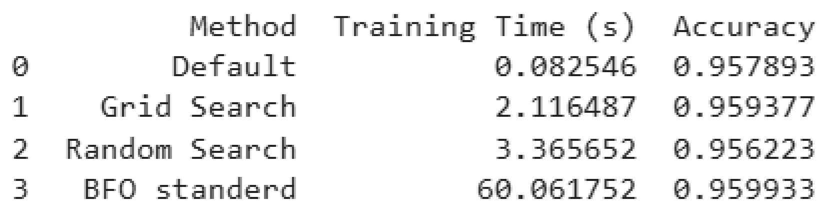


```
 Best Parameters found by BFO:
C = 15.37482
gamma = 0.01370
Best Accuracy (CV) = 0.9599
Training Time = 60.06 seconds
```

Compare SVC Optimizers: Default vs Grid vs Random vs BFO :

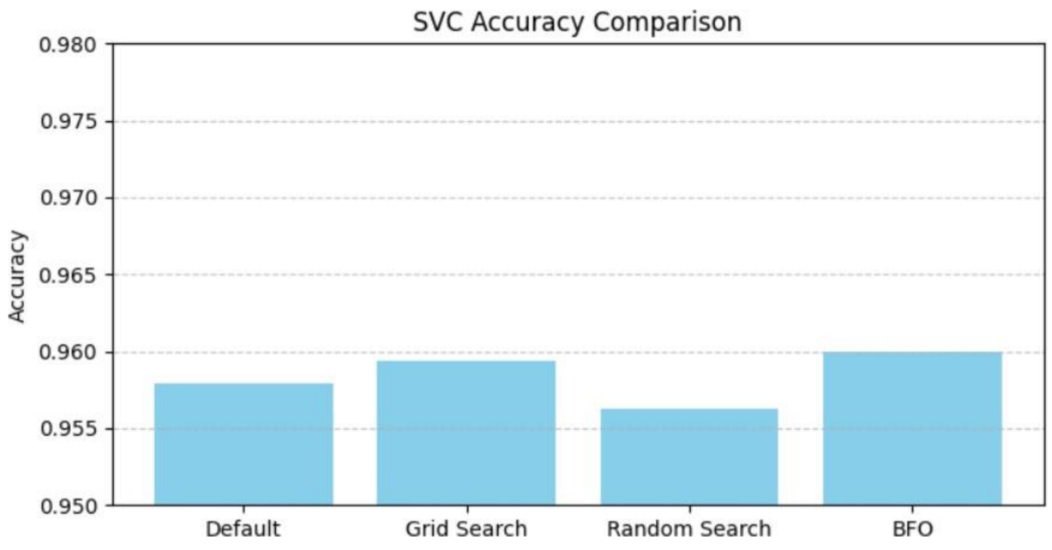
compares four different methods to train and tune an SVC classifier:

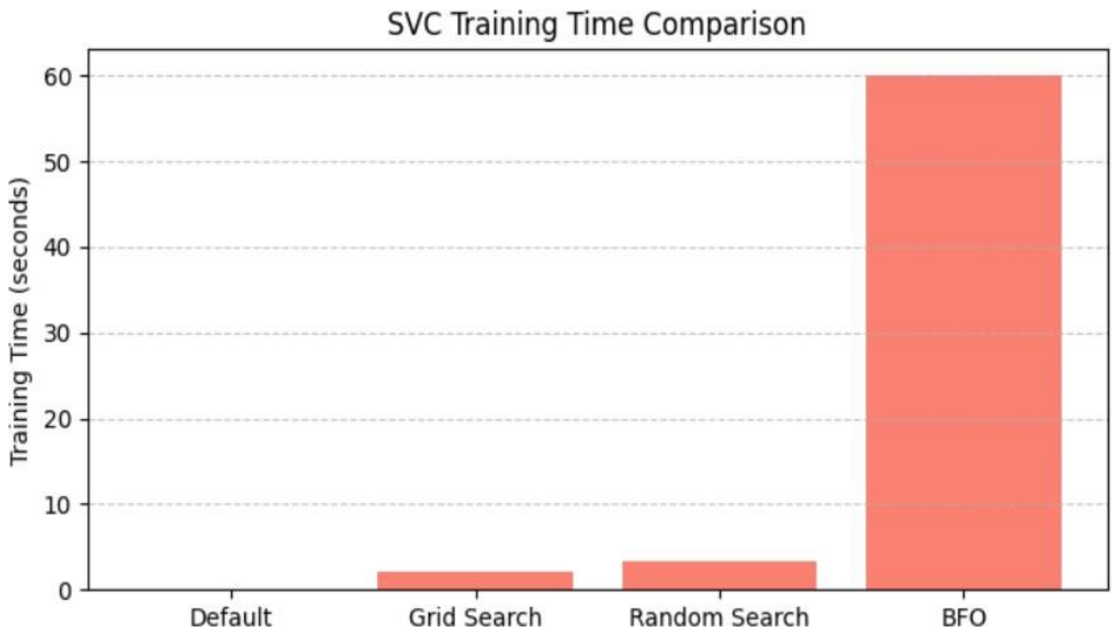
- Default parameters
- Grid Search
- Randomized Search
- Standard BFO



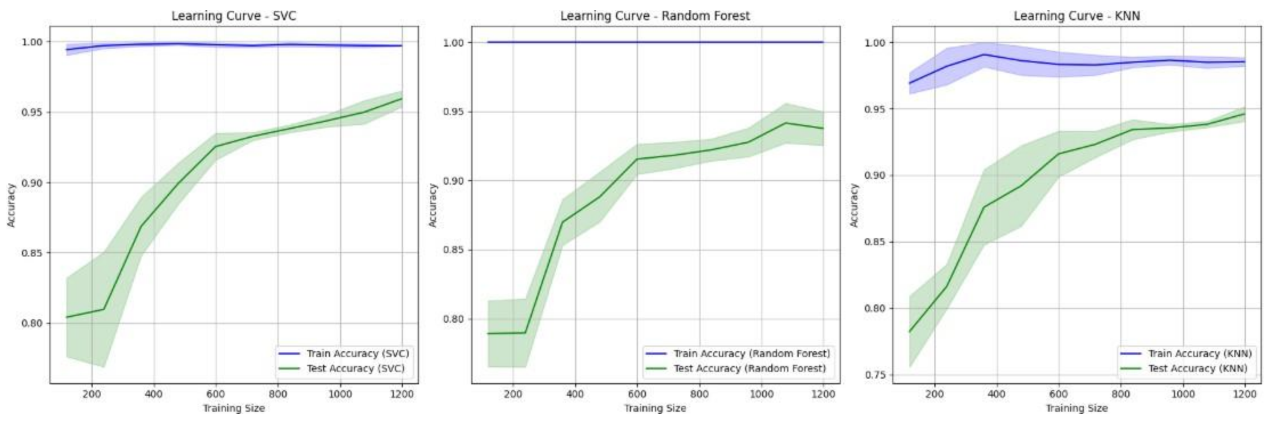
```
          Method  Training Time (s)  Accuracy
0        Default           0.082546  0.957893
1    Grid Search           2.116487  0.959377
2  Random Search           3.365652  0.956223
3   BFO standerd          60.061752  0.959933
```

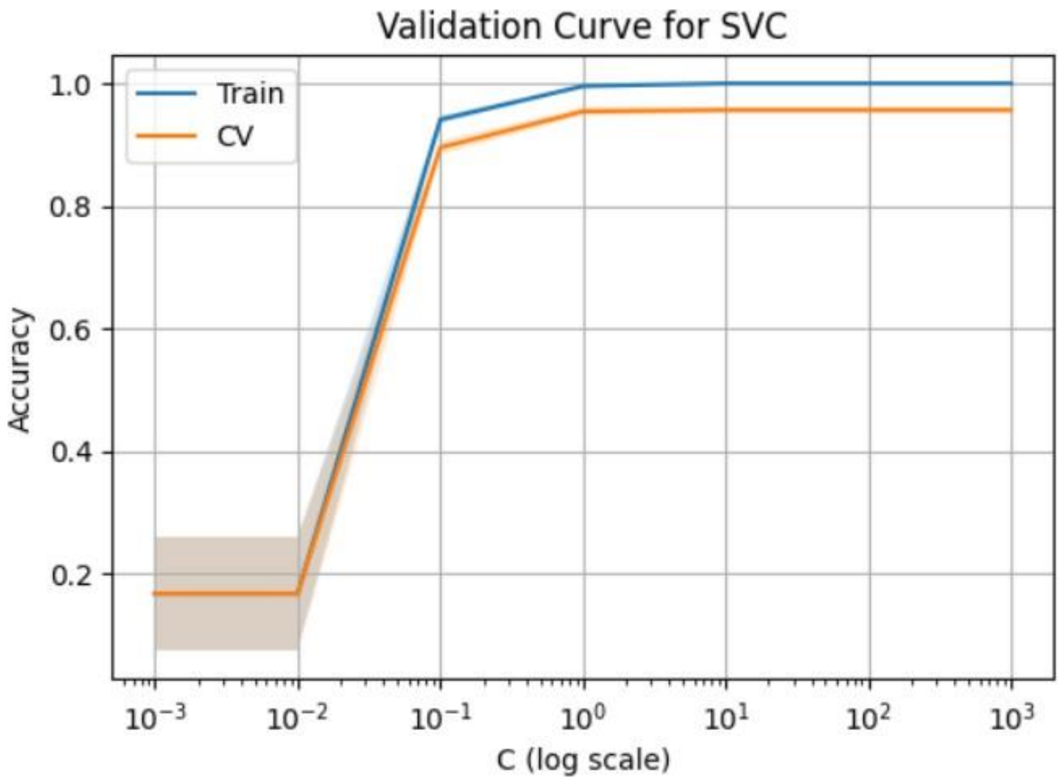
Plotting Accuracy and Training Time Comparisons :

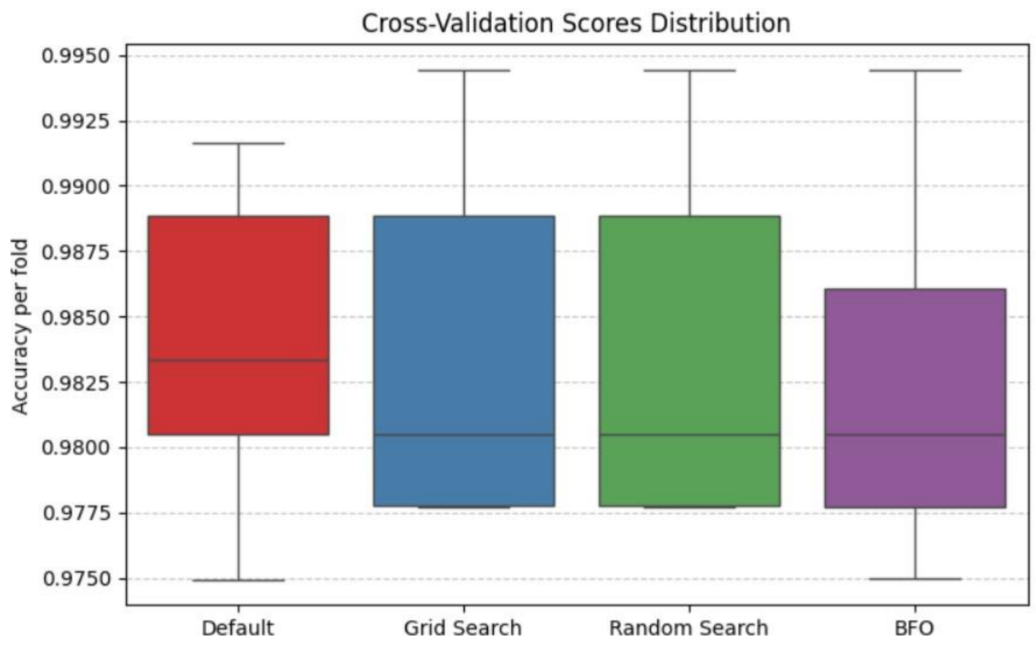




Learning Curve Comparison: SVC

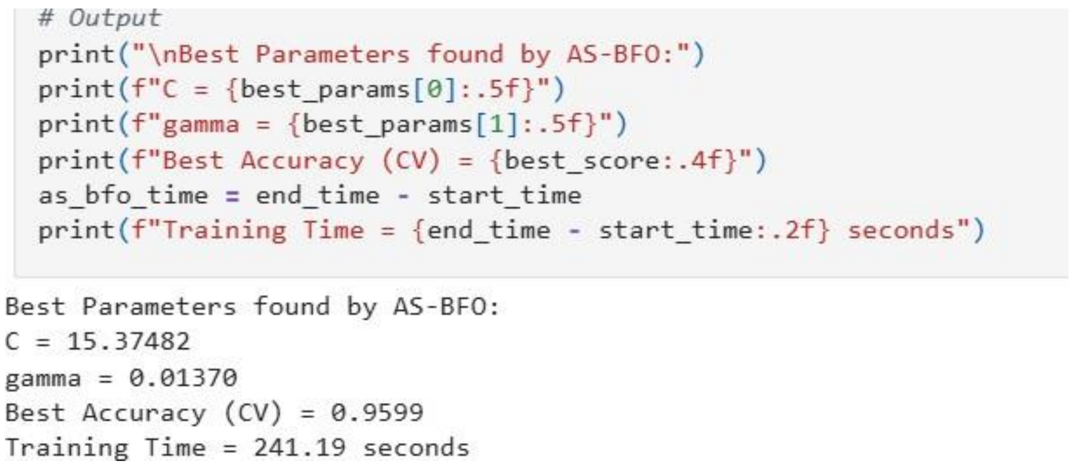





Cross-Validation Scores Distribution
Accuracy per fold
0.9950
0.9925
0.9900
0.9875
0.9850
0.9825
0.9800
0.9775
0.9750
Default
Grid Search
Random Search
BFO

Adaptive BFO Results (C + gamma) :

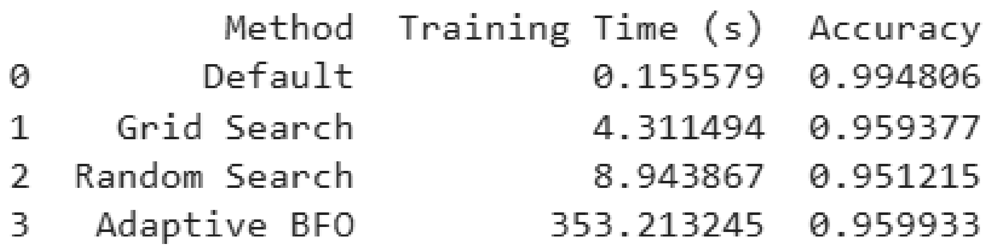
Best parameters found :



```
# Output
print("\nBest Parameters found by AS-BFO:")
print(f"C = {best_params[0]:.5f}")
print(f"gamma = {best_params[1]:.5f}")
print(f"Best Accuracy (CV) = {best_score:.4f}")
as_bfo_time = end_time - start_time
print(f"Training Time = {end_time - start_time:.2f} seconds")
```

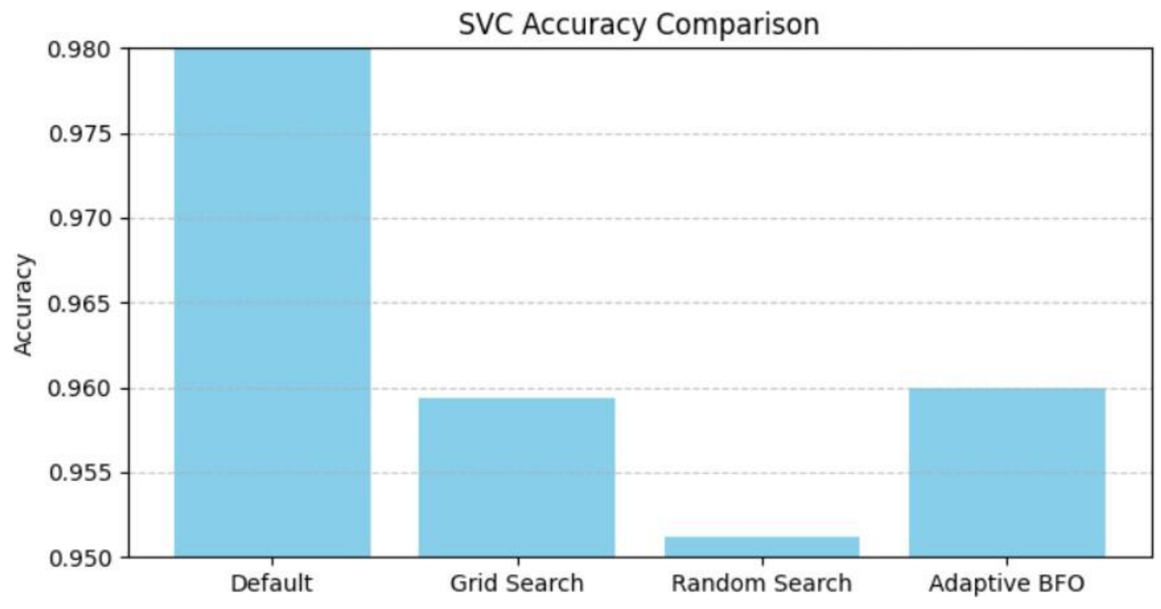
```
Best Parameters found by AS-BFO:
C = 15.37482
gamma = 0.01370
Best Accuracy (CV) = 0.9599
Training Time = 241.19 seconds
```

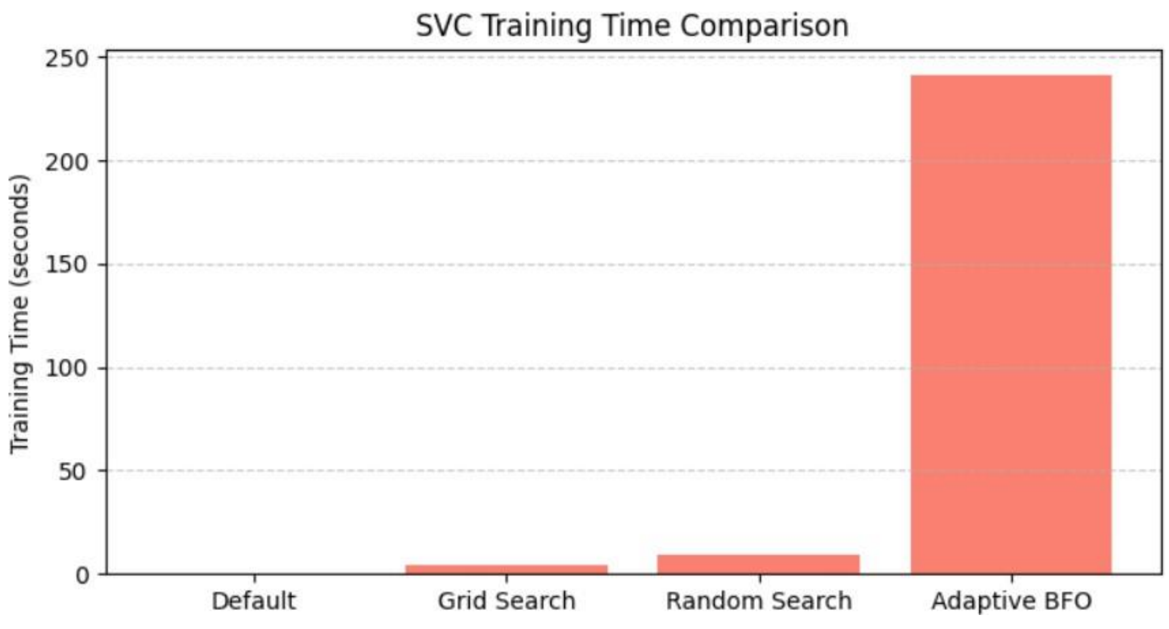
Compare SVC Optimizers: Default vs Grid vs Random vs Adaptive BFO :



```
          Method  Training Time (s)  Accuracy
0        Default           0.155579  0.994806
1    Grid Search           4.311494  0.959377
2  Random Search           8.943867  0.951215
3   Adaptive BFO         353.213245  0.959933
```

Plot Accuracy and Training Time Comparison

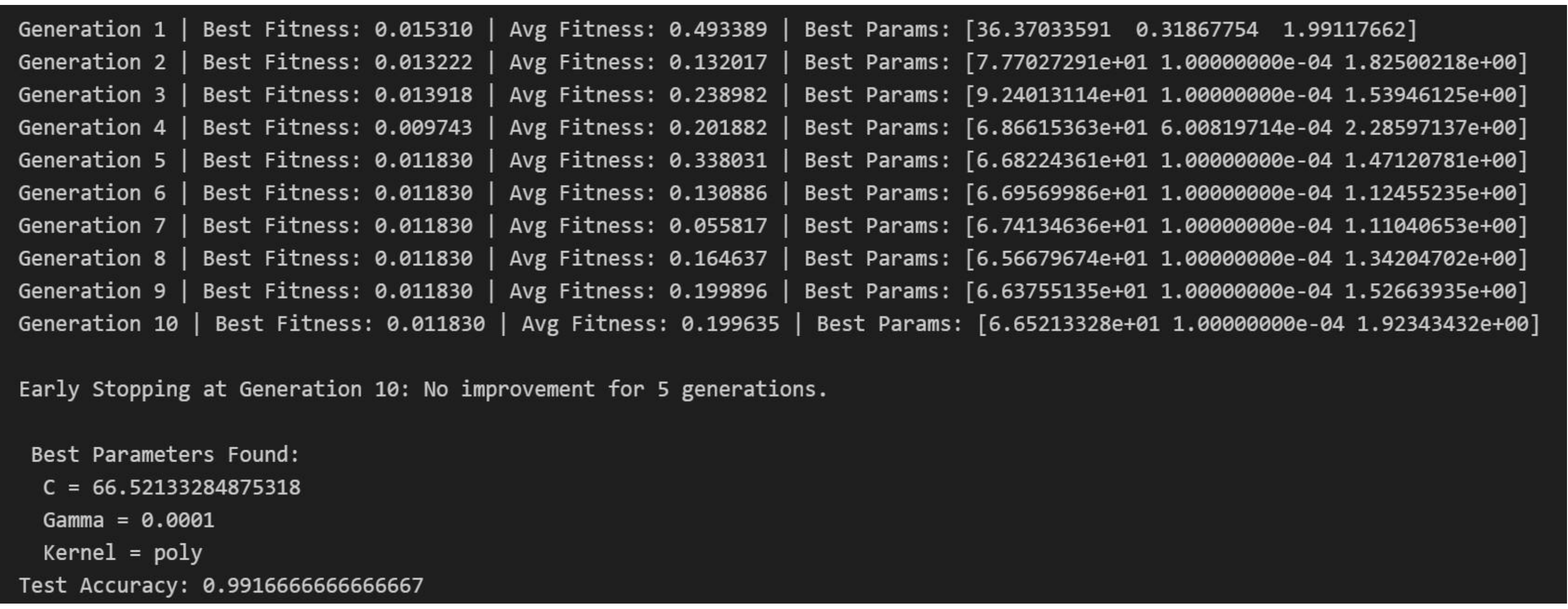




3-Bacterial Foraging Optimization (BFO) with Dynamic Step Size and Mutation :

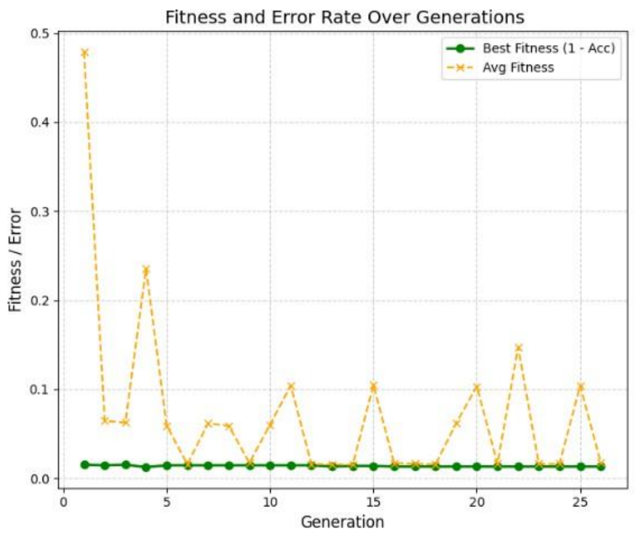
Bacterial Foraging Optimization (BFO) is a bio-inspired metaheuristic algorithm that mimics the foraging behavior of *E. coli* bacteria. The algorithm optimizes solutions by simulating chemotaxis (movement), reproduction, and elimination-dispersal.

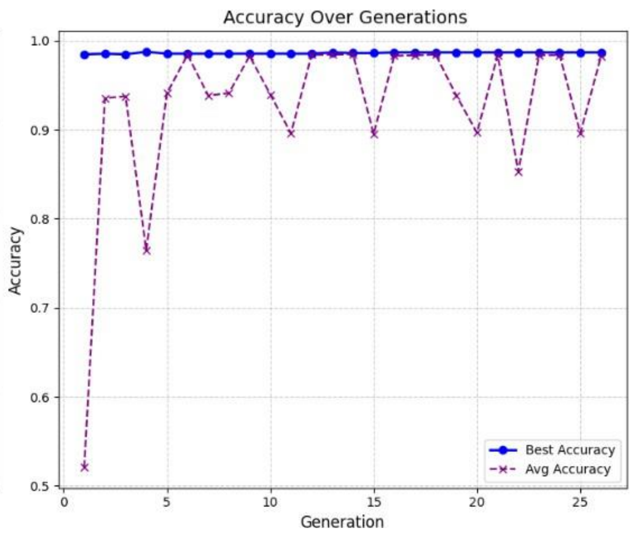
Best Paramters Found during Generations



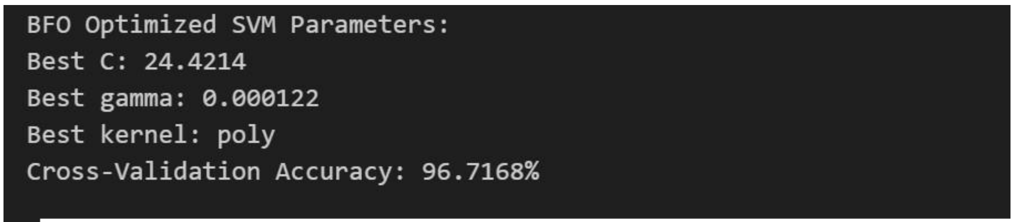
```
Generation 1 | Best Fitness: 0.015310 | Avg Fitness: 0.493389 | Best Params: [36.37033591  0.31867754  1.99117662]
Generation 2 | Best Fitness: 0.013222 | Avg Fitness: 0.132017 | Best Params: [7.77027291e+01 1.00000000e-04 1.82500218e+00]
Generation 3 | Best Fitness: 0.013918 | Avg Fitness: 0.238982 | Best Params: [9.24013114e+01 1.00000000e-04 1.53946125e+00]
Generation 4 | Best Fitness: 0.009743 | Avg Fitness: 0.201882 | Best Params: [6.86615363e+01 6.00819714e-04 2.28597137e+00]
Generation 5 | Best Fitness: 0.011830 | Avg Fitness: 0.338031 | Best Params: [6.68224361e+01 1.00000000e-04 1.47120781e+00]
Generation 6 | Best Fitness: 0.011830 | Avg Fitness: 0.130886 | Best Params: [6.69569986e+01 1.00000000e-04 1.12455235e+00]
Generation 7 | Best Fitness: 0.011830 | Avg Fitness: 0.055817 | Best Params: [6.74134636e+01 1.00000000e-04 1.11040653e+00]
Generation 8 | Best Fitness: 0.011830 | Avg Fitness: 0.164637 | Best Params: [6.56679674e+01 1.00000000e-04 1.34204702e+00]
Generation 9 | Best Fitness: 0.011830 | Avg Fitness: 0.199896 | Best Params: [6.63755135e+01 1.00000000e-04 1.52663935e+00]
Generation 10 | Best Fitness: 0.011830 | Avg Fitness: 0.199635 | Best Params: [6.65213328e+01 1.00000000e-04 1.92343432e+00]

Early Stopping at Generation 10: No improvement for 5 generations.

 Best Parameters Found:
  C = 66.52133284875318
  Gamma = 0.0001
  Kernel = poly
Test Accuracy: 0.9916666666666667
```

Fitness and Error Rate Plot / Accuracy Plot :

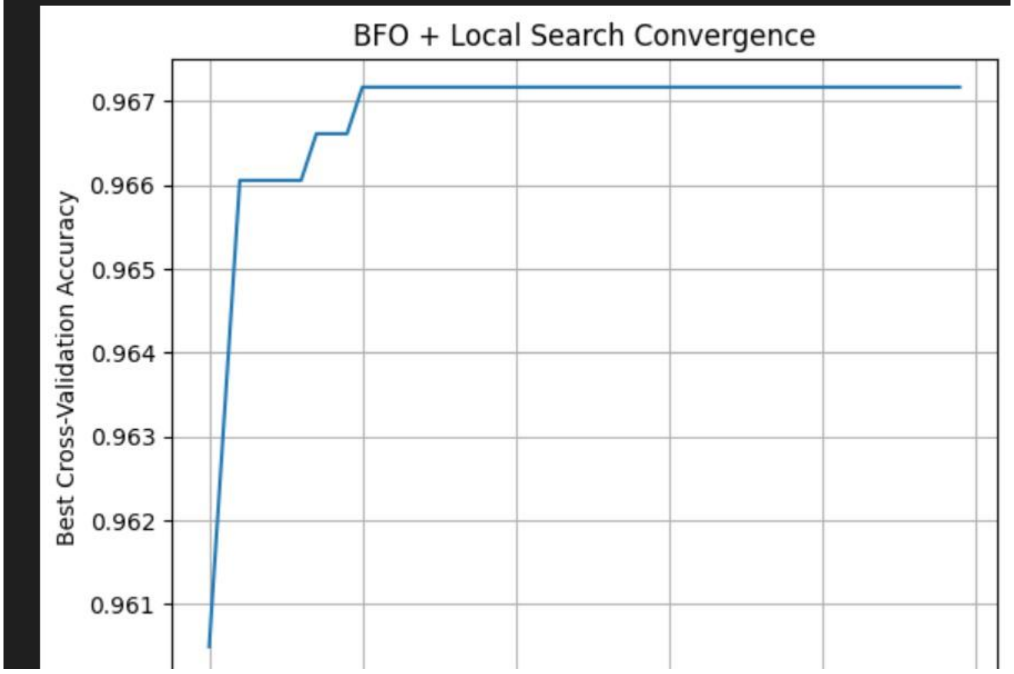




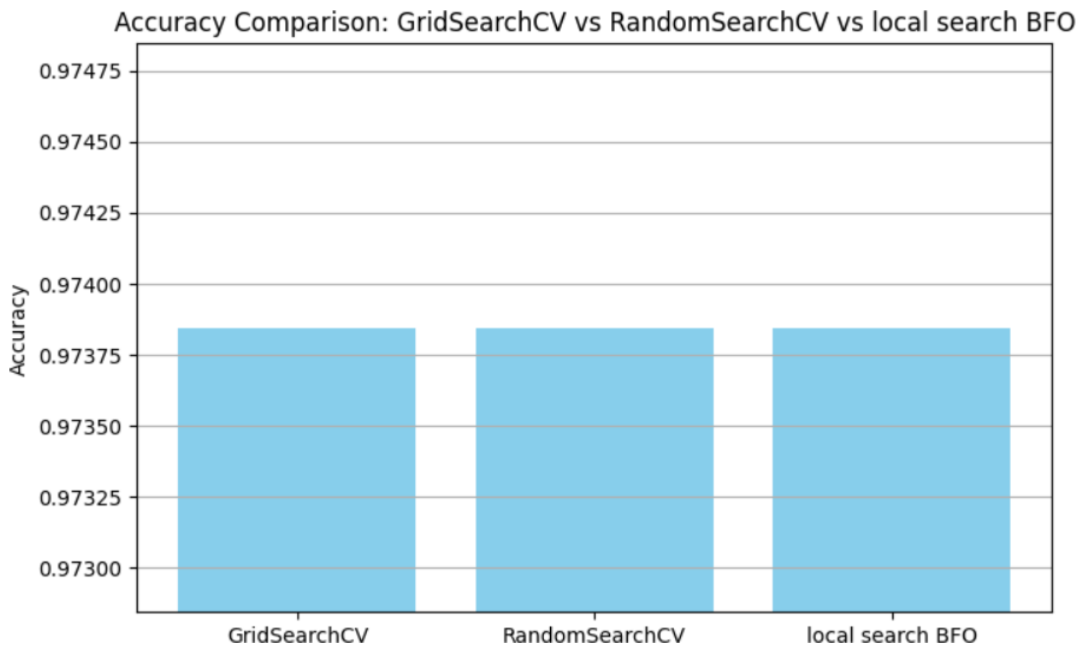
4- Results of BFO with Local Search :



```
BFO Optimized SVM Parameters:
Best C: 24.4214
Best gamma: 0.000122
Best kernel: poly
Cross-Validation Accuracy: 96.7168%
```



Accuracy comparison plot :



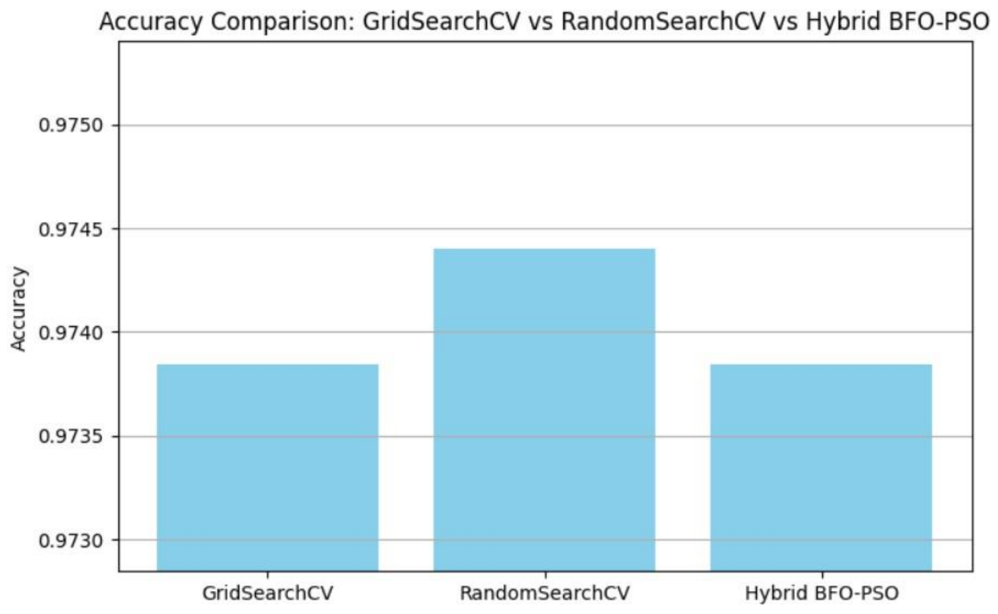
3- Hybrid BFO-PSO optimization Results :

```
print("Best Params (Hybrid BFO-PSO):", best_params_hybrid)
print("Best Accuracy (Hybrid BFO-PSO):", accuracy_hybrid)
```

```
Generation 1 - Best Accuracy: 0.9738
Generation 2 - Best Accuracy: 0.9738
Generation 3 - Best Accuracy: 0.9738
Generation 4 - Best Accuracy: 0.9738
Generation 5 - Best Accuracy: 0.9738
Generation 6 - Best Accuracy: 0.9738
Generation 7 - Best Accuracy: 0.9738
Generation 8 - Best Accuracy: 0.9738
Generation 9 - Best Accuracy: 0.9738
Generation 10 - Best Accuracy: 0.9738
Generation 11 - Best Accuracy: 0.9738
Generation 12 - Best Accuracy: 0.9738
Generation 13 - Best Accuracy: 0.9738
Generation 14 - Best Accuracy: 0.9738
Generation 15 - Best Accuracy: 0.9738
Generation 16 - Best Accuracy: 0.9738
Generation 17 - Best Accuracy: 0.9738
Generation 18 - Best Accuracy: 0.9738
Generation 19 - Best Accuracy: 0.9738
Generation 20 - Best Accuracy: 0.9738
Best Params (Hybrid BFO-PSO): {'C': 68.32131740970121, 'kernel': 'rbf'}
Best Accuracy (Hybrid BFO-PSO): 0.9738452977184195
```

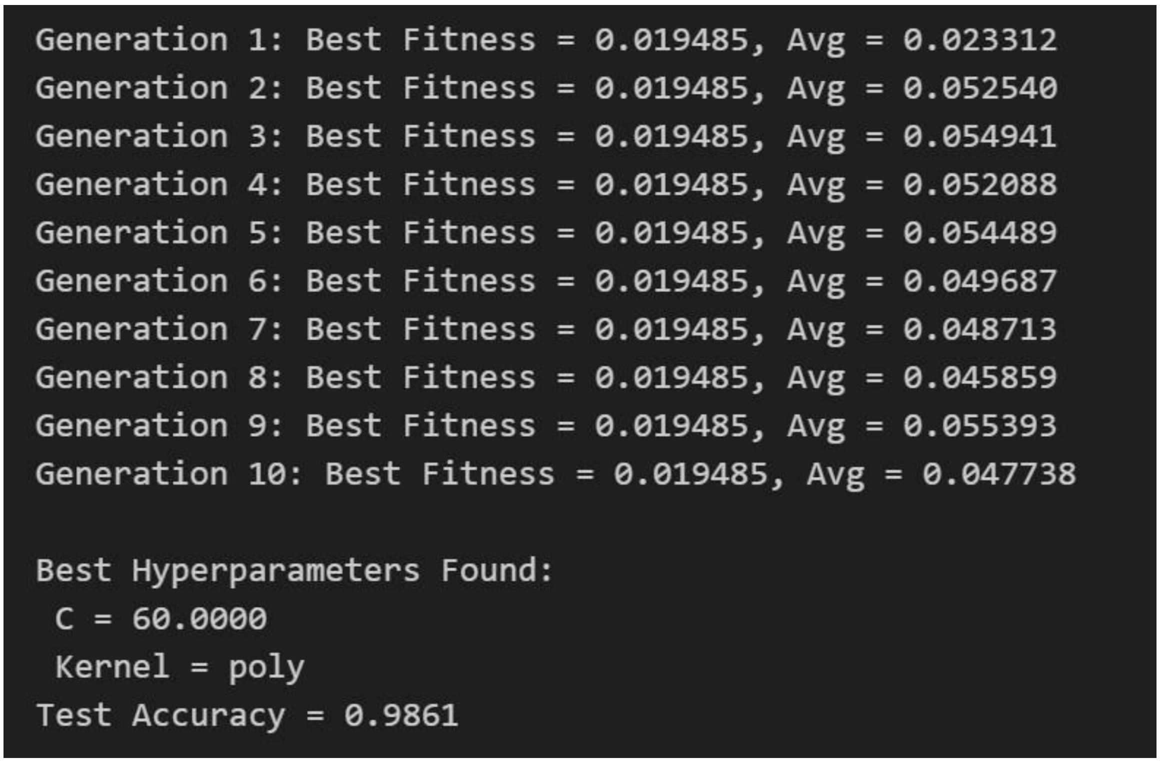
Accuracy Comparison and plot :

```
=== Accuracy Comparison ===
GridSearchCV Accuracy:     0.9738 | Params: {'C': 10, 'kernel': 'rbf'}
RandomSearchCV Accuracy:   0.9744 | Params: {'C': 3.2801676730947085, 'kernel': 'rbf'}
local search BFO Accuracy:      0.9738 | Params: {'C': 68.32131740970121, 'kernel': 'rbf'}
```



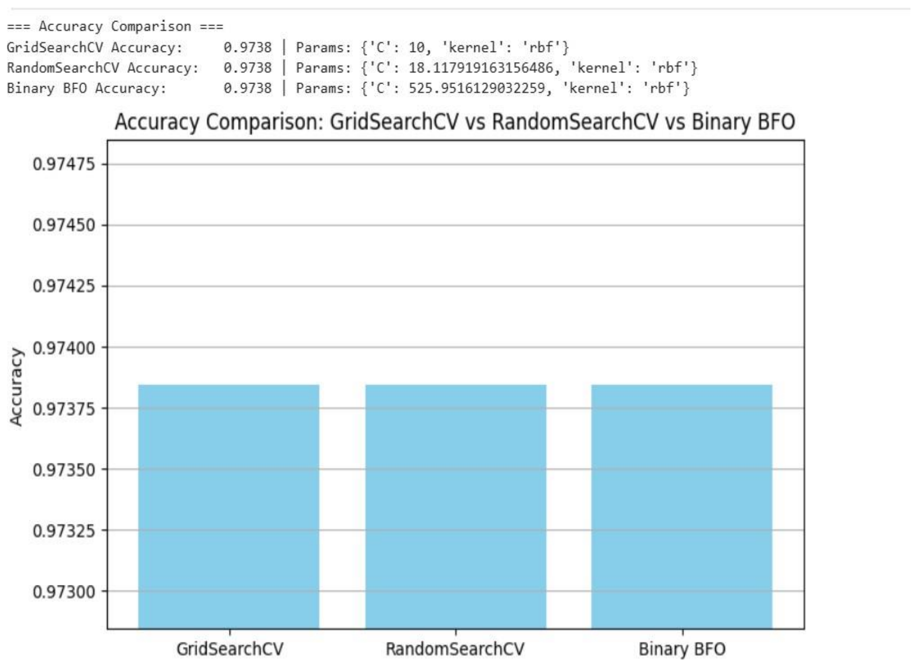
5-Binary BFO Results :

best parameters found during optimization



```
Generation 1: Best Fitness = 0.019485, Avg = 0.023312
Generation 2: Best Fitness = 0.019485, Avg = 0.052540
Generation 3: Best Fitness = 0.019485, Avg = 0.054941
Generation 4: Best Fitness = 0.019485, Avg = 0.052088
Generation 5: Best Fitness = 0.019485, Avg = 0.054489
Generation 6: Best Fitness = 0.019485, Avg = 0.049687
Generation 7: Best Fitness = 0.019485, Avg = 0.048713
Generation 8: Best Fitness = 0.019485, Avg = 0.045859
Generation 9: Best Fitness = 0.019485, Avg = 0.055393
Generation 10: Best Fitness = 0.019485, Avg = 0.047738

Best Hyperparameters Found:
 C = 60.0000
 Kernel = poly
Test Accuracy = 0.9861
```

Accuracy Comparison Plot



```
=== Accuracy Comparison ===
GridSearchCV Accuracy:     0.9738 | Params: {'C': 10, 'kernel': 'rbf'}
RandomSearchCV Accuracy:   0.9738 | Params: {'C': 18.117919163156486, 'kernel': 'rbf'}
Binary BFO Accuracy:       0.9738 | Params: {'C': 525.9516129032259, 'kernel': 'rbf'}
```

performance Evaluation: The performance of each configuration is compared based on classification accuracy and total computational time (HPO plus training). The results are expected to reveal which method achieves the best balance of accuracy and efficiency.

The specific hyperparameters optimized include the regularization parameter C, which controls the trade-off between margin maximization and classification error, and the kernel coefficient gamma, which determines the shape of the decision boundary for the RBF kernel. The BFO algorithms treat these as optimization variables, using a fitness function based on cross-validation accuracy.

Development Platform

The project was developed using Python, leveraging the scikit-learn library (https://scikit-learn.org) for SVM implementation, GridSearchCV, and Random Search. Custom implementations of the BFO algorithms were created to perform hyperparameter optimization, likely using NumPy (https://numpy.org) for numerical computations. The MNIST dataset was loaded using scikit-learn's built-in datasets module. All experiments were conducted in a Jupyter Notebook environment, which facilitates iterative development, visualization, and documentation. This platform ensures reproducibility and ease of use, aligning with standard practices in machine learning research.

Team Members:

The project is led by :

Mohamed Abdelfaataahh
Sara Yasser
Mariam Hany
Omar Tarek
Kerols Emad
Abdelrahman deraz

Conclusion :

This documentation provides a comprehensive overview of the project, focusing on the comparison of traditional and nature-inspired HPO methods for tuning an SVM classifier on the MNIST dataset., the project emphasizes the core aspects of HPO, model training, and evaluation, using accuracy and computational time as the primary metrics. The inclusion of BFO and its variations introduces a novel perspective to HPO, building on academic research that highlights the potential of nature-inspired algorithms. The use of the MNIST dataset ensures reproducibility and comparability, while the Python- based development platform supports efficient implementation. This project contributes to the field of automated machine learning by providing empirical insights into the effectiveness of BFO compared to standard HPO methods, potentially paving the way for further exploration of nature-inspired optimization techniques.