Optimization via Gene Expression Algorithms

Code :

```
import numpy as np

import random

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

# 1. Define the Problem: Create a mathematical function to optimize (Pattern Recognition Task)

# For simplicity, we are using a classification dataset.

def create_synthetic_data():

# Create a simple synthetic classification dataset with 2 classes

X, y = make_classification(n_samples=100, n_features=5, n_classes=2, random_state=42)

return X, y

# 2. Initialize Parameters

population_size = 20

num_genes = 5 # Number of features to use

mutation_rate = 0.1

crossover_rate = 0.7

num_generations = 100

# 3. Initialize Population: Randomly generate genetic sequences

def initialize_population(population_size, num_genes):

population = []

for _ in range(population_size):
```

```
# Randomly initialize each gene between 0 and 1 (binary encoding of features)
genes = np.random.randint(2, size=num_genes)
population.append(genes)
return np.array(population)
# 4. Evaluate Fitness: Based on accuracy of model
def evaluate_fitness(population, X_train, X_test, y_train, y_test):
fitness_scores = []
for individual in population:
# Here, the genes represent feature selection
selected_features = [i for i, gene in enumerate(individual) if gene ==
1] if not selected_features: # if no feature selected, it's an invalid
solution fitness_scores.append(0)
continue
# Train a simple classifier using the selected
features X_train_selected = X_train[:,
selected_features] X_test_selected = X_test[:,
selected_features]
# Train a basic classifier (e.g., Logistic Regression)
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train_selected, y_train)
# Make predictions and calculate accuracy
y_pred = clf.predict(X_test_selected)
accuracy = accuracy_score(y_test, y_pred)
fitness_scores.append(accuracy)
```

```
return np.array(fitness_scores)

# 5. Selection: Tournament Selection

def select_parents(population, fitness_scores):

parents = []

for _ in range(len(population) // 2):

tournament_size = 3

selected = random.sample(list(zip(population, fitness_scores)), tournament_size)

selected = sorted(selected, key=lambda x: x[1], reverse=True)

parents.append(selected[0][0]) # Select the best individual

parents.append(selected[1][0]) # Select the second best individual

return np.array(parents)

# 6. Crossover: Single-point crossover

def crossover(parents):

offspring = []

for i in range(0, len(parents), 2):

parent1 = parents[i]

parent2 = parents[i + 1]

if random.random() < crossover_rate:

crossover_point = random.randint(1, len(parent1) - 1)

child1 = np.concatenate([parent1[:crossover_point],

parent2[crossover_point:]]) child2 =

np.concatenate([parent2[:crossover_point], parent1[crossover_point:]]) else:

child1, child2 = parent1.copy(),

parent2.copy() offspring.append(child1)
```

```
offspring.append(child2)

return np.array(offspring)

# 7. Mutation: Flip bits with mutation rate

def mutate(offspring, mutation_rate):

for i in range(len(offspring)):

for j in range(len(offspring[i])):

if random.random() < mutation_rate:

offspring[i][j] = 1 - offspring[i][j] # Flip the gene

return offspring

# 8. Gene Expression: Decode genetic sequences to functional solutions (feature selection in this case)

# 9. Iterate: Repeat selection, crossover, mutation, and evaluation

def gene_expression_algorithm(X_train, X_test, y_train, y_test, population_size, num_genes, num_generations, mutation_rate, crossover_rate):

population = initialize_population(population_size, num_genes)

for generation in range(num_generations):

fitness_scores = evaluate_fitness(population, X_train, X_test, y_train, y_test)

parents = select_parents(population, fitness_scores)

offspring = crossover(parents)

mutated_offspring = mutate(offspring, mutation_rate)

# Create the new population by replacing the old population with offspring

population = mutated_offspring

# Print the best fitness score for each generation

print(f"Generation {generation + 1}: Best Fitness = {max(fitness_scores)}")

# Return the best solution (individual) from the final population
```

```
final_fitness_scores = evaluate_fitness(population, X_train, X_test, y_train, y_test)

best_individual = population[np.argmax(final_fitness_scores)]

return best_individual

# Main function to run the algorithm with user input

def gwo_pattern_recognition():

# Get user input for generations and population size

generations = int(input("Enter number of generations: "))

population_size = int(input("Enter population size: "))

# Create synthetic data for pattern recognition

X, y = create_synthetic_data()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Run the Gene Expression Algorithm

best_solution = gene_expression_algorithm(X_train, X_test, y_train, y_test, population_size, 5,
generations, 0.1, 0.7)

print(f"Best Feature Selection: {best_solution}")

# Convert best_solution to feature selection

selected_features = [i for i, gene in enumerate(best_solution) if gene == 1]

print(f"Selected Features: {selected_features}")

# Run the program

if __name___== "__main__":

print("Chaitanya N1BM22CS076") # Student Info

gwo_pattern_recognition()
```

Output :

```
Enter number of generations: 50
Enter population size: 20
Generation 1: Best Fitness = 1.0
Generation 2: Best Fitness = 1.0
Generation 3: Best Fitness = 1.0
Generation 4: Best Fitness = 1.0
Generation 5: Best Fitness = 1.0
Generation 6: Best Fitness = 1.0
Generation 7: Best Fitness = 1.0
Generation 8: Best Fitness = 1.0
Generation 9: Best Fitness = 1.0
Generation 10: Best Fitness = 1.0
Generation 11: Best Fitness = 1.0
Generation 12: Best Fitness = 1.0
Generation 13: Best Fitness = 1.0
Generation 14: Best Fitness = 1.0
Generation 15: Best Fitness = 1.0
Generation 16: Best Fitness = 1.0
Generation 17: Best Fitness = 1.0
Generation 18: Best Fitness = 1.0
Generation 19: Best Fitness = 1.0
Generation 20: Best Fitness = 1.0
Generation 21: Best Fitness = 1.0
Generation 22: Best Fitness = 1.0
Generation 23: Best Fitness = 1.0
Generation 24: Best Fitness = 1.0
Generation 25: Best Fitness = 1.0
Generation 26: Best Fitness = 1.0
Generation 27: Best Fitness = 1.0
Generation 28: Best Fitness = 1.0
Generation 29: Best Fitness = 1.0
Generation 30: Best Fitness = 1.0
Generation 31: Best Fitness = 1.0
Generation 32: Best Fitness = 1.0
Generation 33: Best Fitness = 1.0
Generation 34: Best Fitness = 1.0
Generation 35: Best Fitness = 1.0
Generation 36: Best Fitness = 1.0
Generation 37: Best Fitness = 1.0
Generation 38: Best Fitness = 1.0
Generation 39: Best Fitness = 1.0
Generation 40: Best Fitness = 1.0
Generation 41: Best Fitness = 1.0
Generation 42: Best Fitness = 1.0
Generation 43: Best Fitness = 1.0
Generation 44: Best Fitness = 1.0
Generation 45: Best Fitness = 1.0
Generation 46: Best Fitness = 1.0
Generation 47: Best Fitness = 1.0
Generation 48: Best Fitness = 1.0
Generation 49: Best Fitness = 1.0
Generation 50: Best Fitness = 1.0
Best Feature Selection: [1 0 0 1 1]
Selected Features: [0, 3, 4]
```