In [1]:
```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
import random
```

In [2]:
```
df = load_breast_cancer()
X, y = df.data, df.target
```

In [3]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

In [4]:
```
# Parameters
population_size = 10
generations = 20
clone_factor = 3
mutation_rate = 0.1
```

In [5]:
```
# Generate initial population of classifiers
def create_population(size):
    return [DecisionTreeClassifier(max_depth=random.randint(1, 5)) for _ in ra
```

In [6]:
```
# Train and evaluate models to get fitness (affinity)
def evaluate_population(pop):
    affinities = []
    for model in pop:
        model.fit(X_train, y_train)
        pred = model.predict(X_test)
        affinities.append(accuracy_score(y_test, pred))
    return affinities
```

In [7]:
```
# Clone best models
def clone(pop, affinities):
    clones = []
    for i, model in enumerate(pop):
        n_clones = int(affinities[i] * clone_factor * population_size)
        for _ in range(n_clones):
            clones.append(model)
    return clones
```

In [8]:
```
# Apply random mutation (change hyperparameter)
def mutate(clones):
    mutated = []
    for model in clones:
        # Mutate max_depth randomly
        new_depth = max(1, model.get_params()["max_depth"] + random.choice([-1
        mutated_model = DecisionTreeClassifier(max_depth=new_depth)
        mutated.append(mutated_model)
    return mutated
```

In [9]:
```
# Select top models for the next generation
def select_best(pop, affinities, size):
    sorted_pop = [x for _, x in sorted(zip(affinities, pop), key=lambda x: x[0
    return sorted_pop[:size]
```

In [10]:
```
# Main algorithm
population = create_population(population_size)

for gen in range(generations):
    affinities = evaluate_population(population)
    print(f"Generation {gen+1} - Best Affinity: {max(affinities):.4f}")
    clones = clone(population, affinities)
    mutated_clones = mutate(clones)
    all_candidates = population + mutated_clones
    all_affinities = evaluate_population(all_candidates)
    population = select_best(all_candidates, all_affinities, population_size)
```

```
Generation 1 - Best Affinity: 0.9474
Generation 2 - Best Affinity: 0.9474
Generation 3 - Best Affinity: 0.9474
Generation 4 - Best Affinity: 0.9474
Generation 5 - Best Affinity: 0.9474
Generation 6 - Best Affinity: 0.9474
Generation 7 - Best Affinity: 0.9474
Generation 8 - Best Affinity: 0.9474
Generation 9 - Best Affinity: 0.9474
Generation 10 - Best Affinity: 0.9474
Generation 11 - Best Affinity: 0.9474
Generation 12 - Best Affinity: 0.9474
Generation 13 - Best Affinity: 0.9474
Generation 14 - Best Affinity: 0.9474
Generation 15 - Best Affinity: 0.9474
Generation 16 - Best Affinity: 0.9474
Generation 17 - Best Affinity: 0.9474
Generation 18 - Best Affinity: 0.9474
Generation 19 - Best Affinity: 0.9474
Generation 20 - Best Affinity: 0.9474
```

In [11]:
```
# Final evaluation
final_model = population[0]
final_model.fit(X_train, y_train)
y_pred = final_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"\nFinal Test Accuracy: {accuracy * 100:.2f}%")
```

```

Final Test Accuracy: 94.74%
```

In [ ]: