Machine Learning End Lab Exam Question 2

Name: Sri Sai Vijaya Aditya Nittala
Roll No.: 177163
Section: A

Classify the above boolean function using the GABIL method (Genetic Algorithm):

> Y = (A + AB)(B + BC)(C + AB)

This boolean function can be simplified as follows:

> Y = (A + AB)(B + BC)(C + AB) = (A)(B)(C + AB) = ABC + (AB)(AB) = ABC + AB = AB(C + 1) = AB

Now, this boolean function is nothing but an AND function\

Using genetic algorithm we are trying to optimize the weights given the target function. The aim is to converge the weights such that classification is optimal.

```
## IMPORTING REQUIRED LIBRARIES
import numpy as np
import random


## TRUTH TABLE FOR 3 BOOLEAN VARIABLES
X = [1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1
X = np.reshape(X, (8, 4))

## -1 INDICATES 0 AND 1 INDICATES 1
y = [-1, -1, -1, -1, -1, -1, 1, 1]
y = np.reshape(y, (8, 1))


## CREATING INITIAL POPULATION
no_output = 8
num_weights = 4
population_size = (no_output, num_weights)

new_population = np.random.uniform(low = -5, high = 5, size = population_size)


## FITNESS FUNCTION
def Fitness(X, y, population):
    fitness = np.zeros(len(population))
    for i in range(len(population)):
        e = 0
```

```
        for j in range(len(X)):
            e = e + np.square(np.dot(X[j], population[i]) - y[j])
        e = e/len(X)
        fitness[i] = e

    return fitness


## SELECTING BEST INDIVIDUALS OF CURRENT GENERATION TO BECOME PARENTS OF NEXT GENERATION
def SelectMatingPool(population, fitness, num_parents):
    parents = np.empty((num_parents, population.shape[1]))
    parent_idx = []
    for p in range(num_parents):
        pos = np.argmin(fitness)
        parent_idx.append(pos)
        parents[p, :] = population[pos, :]
        fitness[pos] = np.inf

    return parents


## CROSSING OVER GENES INDICATES NEW GENERATION IS BEING FORMED
def Crossover(parents, offspring_size):
    os = np.empty(offspring_size)
    cp = np.uint8(offspring_size[1]/2)

    for k in range(offspring_size[0]):
        p1 = k%parents.shape[0]
        p2 = (k+1)%parents.shape[0]
        os[k, 0:cp] = parents[p1, 0:cp] ## first half from p1 and second half from p2
        os[k, cp:] = parents[p2, cp:] ## opposite of above

    return os


## SOME OF THE NEW MEMBERS OF THE POPULATION ARE MUTATED RANDOMLY
def Mutation(os):
    for i in range(os.shape[0]):
        rv = np.random.uniform(-1, 1, 1)
        val = np.random.randint(os.shape[1])
        os[i, val] = os[i, val] + rv

    return os


'''
    For running the GA algorithm, the following steps are:
    1. Generate population
    2. Calculate Fitness
    3. Select fittest individuals for crossover
    4. Crossover
    5. Mutate
    6. New population generated
```

```
'''
generations = 5
parents_mating = 4

for g in range(generations):
    fitness = Fitness(X, y, new_population)
    parents = SelectMatingPool(new_population, fitness, parents_mating)
    osc = Crossover(parents, offspring_size=(population_size[0] - parents.shape[0], num_weigh
    osm = Mutation(osc)

    new_population[0:parents.shape[0], :] = parents
    new_population[parents.shape[0]:, :] = osm


weights = new_population[np.argmin(fitness)]
print("Final Weights are : {}".format(weights))
```

```
Final Weights are : [-3.13436398  4.48189661  1.23189533 -1.04659033]
```

```
print("Truth table for the given boolean function : ")
print("A\tB\tC\tY\tH")
print("-----"*7)
for i in range(X.shape[0]):
    for j in range(1, X.shape[1]):
        print("{}\t".format(X[i, j]), end = '')
    print("{}\t".format(0 if y[i, 0] == -1 else 1), end='')
    print('{}'.format(0 if np.dot(X[i], weights) < 0.5 else 1))
```

```
Truth table for the given boolean function :
A       B       C       Y       H
-----------------------------------
0       0       0       0       0
0       0       1       0       0
0       1       0       0       0
0       1       1       0       0
1       0       0       0       1
1       0       1       0       0
1       1       0       1       1
1       1       1       1       1
```