**MACHINE LEARNING & DEEP LEARNING**

NAME: MONIKA BHAGWAN DESAI

ROLL NO: 531

MSC-CS(SEM 3)

**ASSIGNMENT: Implement Genetic Algorithm with three Operator.**

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics.

**They are commonly used to generate high-quality solutions for optimization problems and search problems.**

**Operators of Genetic Algorithms**

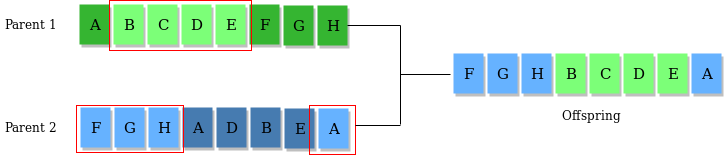
**1) Selection Operator:**

The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations. 

**2) Crossover Operator:**

This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).

For example –



**3) Mutation Operator:**

 The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence.

For Example:



**Example Problem:**

Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made –

* Characters A-Z, a-z, 0-9, and other special symbols are considered as genes
* A string generated by these characters is considered as chromosome/solution/Individual

Fitness score: This is the number of characters which differ from characters in target string at a particular index. So individual having lower fitness value is given more preference.

CODE:

import random

# Number of individuals in each generation

POPULATION\_SIZE = 100

# Valid genes

GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP

QRSTUVWXYZ 1234567890, .-;:\_!"#%&/()=?@${[]}'''

# Target string to be generated

TARGET = "I love India"

class Individual(object):

'''

Class representing individual in population

'''

def \_\_init\_\_(self, chromosome):

self.chromosome = chromosome

self.fitness = self.cal\_fitness()

@classmethod

def mutated\_genes(self):

'''

create random genes for mutation

'''

global GENES

gene = random.choice(GENES)

return gene

@classmethod

def create\_gnome(self):

'''

create chromosome or string of genes

'''

global TARGET

gnome\_len = len(TARGET)

return [self.mutated\_genes() for \_ in range(gnome\_len)]

def mate(self, par2):

'''

Perform mating and produce new offspring

'''

# chromosome for offspring

child\_chromosome = []

for gp1, gp2 in zip(self.chromosome, par2.chromosome):

# random probability

prob = random.random()

# if prob is less than 0.45, insert gene

# from parent 1

if prob < 0.45:

child\_chromosome.append(gp1)

# if prob is between 0.45 and 0.90, insert

# gene from parent 2

elif prob < 0.90:

child\_chromosome.append(gp2)

# otherwise insert random gene(mutate),

# for maintaining diversity

else:

child\_chromosome.append(self.mutated\_genes())

# create new Individual(offspring) using

# generated chromosome for offspring

return Individual(child\_chromosome)

def cal\_fitness(self):

'''

Calculate fitness score, it is the number of

characters in string which differ from target

string.

'''

global TARGET

fitness = 0

for gs, gt in zip(self.chromosome, TARGET):

if gs != gt: fitness+= 1

return fitness

# Driver code

def main():

global POPULATION\_SIZE

#current generation

generation = 1

found = False

population = []

# create initial population

for \_ in range(POPULATION\_SIZE):

gnome = Individual.create\_gnome()

population.append(Individual(gnome))

while not found:

# sort the population in increasing order of fitness score

population = sorted(population, key = lambda x:x.fitness)

# if the individual having lowest fitness score ie.

# 0 then we know that we have reached to the target

# and break the loop

if population[0].fitness <= 0:

found = True

break

# Otherwise generate new offsprings for new generation

new\_generation = []

# Perform Elitism, that mean 10% of fittest population

# goes to the next generation

s = int((10\*POPULATION\_SIZE)/100)

new\_generation.extend(population[:s])

# From 50% of fittest population, Individuals

# will mate to produce offspring

s = int((90\*POPULATION\_SIZE)/100)

for \_ in range(s):

parent1 = random.choice(population[:50])

parent2 = random.choice(population[:50])

child = parent1.mate(parent2)

new\_generation.append(child)

population = new\_generation

print("Generation: {}\tString: {}\tFitness: {}".\

format(generation,

"".join(population[0].chromosome),

population[0].fitness))

generation += 1

print("Generation: {}\tString: {}\tFitness: {}".\

format(generation,

"".join(population[0].chromosome),

population[0].fitness))

if \_\_name\_\_ == '\_\_main\_\_':

main()

OUTPUT:

Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

===================== RESTART: C:/Users/Admin/Desktop/p1.py ====================

Generation: 1 String: fKFx#G( FhiJ Fitness: 11

Generation: 2 String: 7?l4:T=a1d ? Fitness: 10

Generation: 3 String: 7?l4:T=a1d ? Fitness: 10

Generation: 4 String: b k4vGa FhiJ Fitness: 9

Generation: 5 String: .sloI[H 1hia Fitness: 8

Generation: 6 String: .sloI[H 1hia Fitness: 8

Generation: 7 String: lbvr nvRia Fitness: 6

Generation: 8 String: lbvr nvRia Fitness: 6

Generation: 9 String: I lPv1=(1dia Fitness: 5

Generation: 10 String: I lPv1=(1dia Fitness: 5

Generation: 11 String: I lPv1 (rdia Fitness: 4

Generation: 12 String: I lbv1 nndia Fitness: 3

Generation: 13 String: I lbv1 nndia Fitness: 3

Generation: 14 String: I lbv1 nndia Fitness: 3

Generation: 15 String: I lov1 (ndia Fitness: 2

Generation: 16 String: I lov1 (ndia Fitness: 2

Generation: 17 String: I lov1 (ndia Fitness: 2

Generation: 18 String: I lov1 (ndia Fitness: 2

Generation: 19 String: I lov1 (ndia Fitness: 2

Generation: 20 String: I lov1 (ndia Fitness: 2

Generation: 21 String: I lov1 (ndia Fitness: 2

Generation: 22 String: I lov1 (ndia Fitness: 2

Generation: 23 String: I lov1 (ndia Fitness: 2

Generation: 24 String: I lov1 (ndia Fitness: 2

Generation: 25 String: I lov1 (ndia Fitness: 2

Generation: 26 String: I lov1 (ndia Fitness: 2

Generation: 27 String: I lov1 (ndia Fitness: 2

Generation: 28 String: I lov1 (ndia Fitness: 2

Generation: 29 String: I lov1 (ndia Fitness: 2

Generation: 30 String: I lov1 (ndia Fitness: 2

Generation: 31 String: I lov1 (ndia Fitness: 2

Generation: 32 String: I lov1 (ndia Fitness: 2

Generation: 33 String: I lov1 (ndia Fitness: 2

Generation: 34 String: I lov1 (ndia Fitness: 2

Generation: 35 String: I lov1 (ndia Fitness: 2

Generation: 36 String: I lov1 (ndia Fitness: 2

Generation: 37 String: I lovq India Fitness: 1

Generation: 38 String: I lovq India Fitness: 1

Generation: 39 String: I lovq India Fitness: 1

Generation: 40 String: I lovq India Fitness: 1

Generation: 41 String: I lovq India Fitness: 1

Generation: 42 String: I lovq India Fitness: 1

Generation: 43 String: I lovq India Fitness: 1

Generation: 44 String: I lovq India Fitness: 1

Generation: 45 String: I lovq India Fitness: 1

Generation: 46 String: I love India Fitness: 0

