```python
# @Name: Pratik Walawalkar
# @UIN: 667624808

# importing libraries: random and matplotlib
# random: for generating random numbers, matplotlib: for plotting graphs
import random
import matplotlib.pyplot as plt

# generating optimal weights i.e. w randomly between mentioned range
w0 = random.uniform(-0.25,0.25)
w1 = random.uniform(-1,1)
w2 = random.uniform(-1,1)
w = []
w.append(w0)
w.append(w1)
w.append(w2)

# generating w' randomly between mentioned range for carrying out PTA
wi0 = random.uniform(-1,1)
wi1 = random.uniform(-1,1)
wi2 = random.uniform(-1,1)
wi = []
wi.append(wi0)
wi.append(wi1)
wi.append(wi2)

# function for perceptron training algorithm
def PTA(experiment):

    print("Optimal weights before PTA i.e. w: ",w)
    print("Updated weights for carrying out PTA i.e. w': ",wi)
    T = []
    s = []
    s0 = []
    s1 = []
    d = []

# picking S= x1,...,xn vectors indepedently and uniformly at random in [-1,1]
# creating [1 x1 x2]  list corresponding to S collection
    for p in range(experiment):
            T.append([])
            T[p].append(1)
            T[p].append(random.uniform(-1,1))
            T[p].append(random.uniform(-1,1))

# matrix multiplication - [1 x1 x2][w0 w1 w2]T with step activation function as u(.)
    for i in range(experiment):

            sum= (w[0]*T[i][0])+(w[1]*T[i][1])+(w[2]*T[i][2])
            T[i].pop(0)
            s.append(T[i])
            if sum<0:
                d.append(0)    # collection of desired outputs = 0
                s0.append(T[i])  # collection of S0 vectors where S0 is subset of S
            else:
                d.append(1)     # collection of desired outputs = 1
                s1.append(T[i])  # collection of S1 vectors where S1 is subset of S
# d is collection of all desired outputs containing zeros and ones

#    print("S: ",s,"\r\nTotal vectors in S: ",len(s))
#    print("\r\nTotal vectors in S0: ",len(s0))
#    print("\r\nTotal vectors in S1: ",len(s1))

# plotting S1 and S0 collection of (x1,x2) vectors
    for x in s1:
        x1 = x[0]
        x2 = x[1]
        plt.plot(x1,x2,'gs')

    for x in s0:
        x1 = x[0]
        x2 = x[1]
```

```
        plt.plot(x1,x2,'ro')

# plotting line: w0 + w1x1 + w2x2 = 0
    m1 = (-w0-w2)/w1 # x-intercept for x1
    m2 = (-w0+w2)/w1 # x-intercept for x2
    xline = [m1,m2]
    yline = [1,-1]
    plt.title("Plot before PTA")
    plt.axis([-1, 1, -1, 1])
    plt.plot(xline,yline,'b-',lw=2)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.show()

# red circles denote collection of S0 vectors(<0), green squares denote collection of S1 vectors(>=0)
    print("\r\nIndex-> Red = Class S0 ; Green = Class S1\r\n")

# creating Ti collection vectors of [1 x1 x2]
    Ti = []
    for e in range(experiment):
        Ti.append([])
        Ti[e] = [1]+s[e]

    eta = [1,0.1,10] # training parameters (1, 0.1, 10)
    for n in range(3):
        misarray = []
        epocharray = []
        epoch = 0
        flag = 0
        sum = 0
        wii = []
# creating local copy of w' and storing in new weight variable (used for updating after every misclassfication) i.e. wu
        wu0 = wi0
        wu1 = wi1
        wu2 = wi2
        while(flag==0):
            mis = 0 # counter for misclassfications
# Perceptron Training Algorithm
            for e in range(experiment):
# matrix multiplication of [1 x1 x2] and wu weights
                sum = (wu0*Ti[e][0])+(wu1*Ti[e][1])+(wu2*Ti[e][2])
                if sum<0:
                    cal = 0 # if above result is less than 0, calculated o/p = 0
                else:
                    cal = 1 # if above result is greater than or equal to 0, calculated o/p = 1
                if (cal != d[e]):
                    mis = mis + 1 # if calculated o/p not equal to desired output for S(x1,x2) vector, then increment misclassfication
# update weights as per PTA if there is misclassfication
                    wu0 = wu0 + (eta[n]*(Ti[e][0])*(d[e]-cal))
                    wu1 = wu1 + (eta[n]*(Ti[e][1])*(d[e]-cal))
                    wu2 = wu2 + (eta[n]*(Ti[e][2])*(d[e]-cal))
            wii.append([])
            wii[epoch].append(wu0)
            wii[epoch].append(wu1)
            wii[epoch].append(wu2)
            epoch = epoch+1 #increment epoch number when all samples are fed in PTA
            misarray.append(mis) # array for getting range for misclassfication for plotting graph
            epocharray.append(epoch) # array for getting range for epoch for plotting graph

            if mis==0:
                flag = 1 # If misclassfication is zero that means our PTA has completed succesffully and comes out of while loop
            else:
                flag = 0

        print("Weights after first epoch i.e. w'' : ",wii[0]) # w'' after first epoch
```

```
138           print("For eta = ",eta[n]," :")
139           print("Total number of epochs required for convergence: ",epoch) # Total
              number of epochs
140           print("Final weights: ",wii[epoch-1]) # Final weights where convergence is
              achieved
141
142   # plot for epoch v/s misclassfication for each eta and samples(100 and 1000)
143           plt.title("Epoch V/S Miss")
144           plt.axis([0,epoch+1,0,100])
145           plt.plot(epocharray,misarray,'o-')
146           plt.show()
147
148       return
149
150   print("------------Perceptron Training Algorithm with 100
      samples-----------------\r\n")
151   # passing 100 samples into PTA function
152   PTA(100)
153   print("------------Perceptron Training Algorithm with 1000
      samples----------------\r\n")
154   # passing 1000 samples into PTA function
155   PTA(1000)
156
157
158
159
```