Ahsanullah University of Science & Technology

Department of Computer Science & Engineering

Course Title : Pattern Recognition Lab

Course No : CSE 4214

Assignment No : 01

Date of Submission : 07/06/2023

Submitted To : Mr. Sajib Kumar Saha, Mr. Faisal Muhammad Shah

Submitted By-

Group : A_2

Name : Ahmadul Karim Chowdhury

Id : 190104037

Section : A

- Course No : CSE 4214
- Course Name: Pattern Recognition Lab
- Name: Ahmadul Karim Chowdhury
- ID: 190104037
- Group: A2
- Experiment No: 02
- Experiment Name: Implementing the Perceptron algorithm for finding the weights * of a Linear Discriminant function.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

```
%cd drive/MyDrive/Colab Notebooks/CSE 4214 Pattern Recognition Lab/
```

```
/content/drive/MyDrive/Colab Notebooks/CSE 4214 Pattern Recognition Lab
```

```
!ls
```

```
'190104037_Assignment 01.ipynb'   PR_Lab2.ipynb               train.txt
 pandas_lab.ipynb                 'PR_Lab2 (Practice).ipynb'
'pandas_lab (Practice).ipynb'     tips.csv
```

Task 1

Take input from “train.txt” file.

```
df = pd.read_csv('train.txt', sep=" ", header=None, dtype='float64')
print(df)

np_train_arr = df.to_numpy()
```

```
      0    1    2
0   2.0  2.0  1.0
1   3.0  1.0  1.0
2  -4.0  3.0  2.0
3   3.0  3.0  1.0
4  -1.0 -3.0  1.0
5   2.0  6.0  2.0
6   4.0  2.0  1.0
7  -2.0 -2.0  1.0
8   0.0  0.0  2.0
9  -2.0  2.0  2.0
10 -1.0 -1.0  2.0
11 -4.0  2.0  2.0
```

```
train_X = df.iloc[:, 0:2]
train_Y = df.iloc[:, 2]
train_X = np.array(train_X)
train_Y = np.array(train_Y)

print(train_X)
print(train_Y)
```

```
[[ 2.  2.]
 [ 3.  1.]
 [-4.  3.]
 [ 3.  3.]
 [-1. -3.]
 [ 2.  6.]
 [ 4.  2.]
 [-2. -2.]
 [ 0.  0.]
 [-2.  2.]
 [-1. -1.]
 [-4.  2.]]
[1. 1. 2. 1. 1. 2. 1. 1. 2. 2. 2. 2.]
```

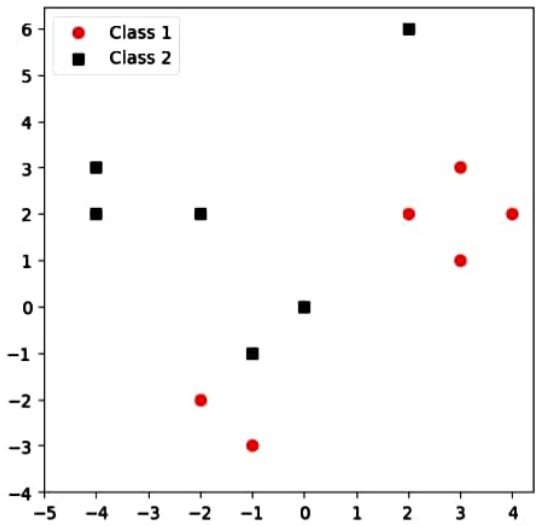
Splitting training data into classes according to labels

```
class1_X1, class1_X2, class2_X1, class2_X2 = [], [], [], []

for i in range(train_X.shape[0]):
  if train_Y[i] == 1:
    class1_X1.append(train_X[i, 0])
    class1_X2.append(train_X[i, 1])
  else:
    class2_X1.append(train_X[i, 0])
    class2_X2.append(train_X[i, 1])

print(class1_X1)
print(class1_X2)
print(class2_X1)
print(class2_X2)
```

```
[2.0, 3.0, 3.0, -1.0, 4.0, -2.0]
[2.0, 1.0, 3.0, -3.0, 2.0, -2.0]
[-4.0, 2.0, 0.0, -2.0, -1.0, -4.0]
[3.0, 6.0, 0.0, 2.0, -1.0, 2.0]
```

Plot all sample points from both classes

```
limit_X1 = class1_X1 + class2_X1
limit_X2 = class1_X2 + class2_X2

plt.figure(figsize=(5, 5))
# Training Data
plt.scatter(class1_X1, class1_X2, label='Class 1', color='red', marker='o')
plt.scatter(class2_X1, class2_X2, label='Class 2', color='black', marker='s')
# Plot Accessory
plt.xticks(range(int(min(limit_X1)) - 1, int(max(limit_X1)) + 1))
plt.yticks(range(int(min(limit_X2)) - 1, int(max(limit_X2)) + 1))
plt.legend(loc='upper left')
plt.show()

```



Task 2

Generate the high dimensional sample points y using formula

```
trainPoints = np.zeros((train_X.shape[0], 6))

j = len(class1_X1)

for i in range(j):
  trainPoints[i, :] = np.array([class1_X1[i] ** 2, class1_X2[i] ** 2,
                                class1_X1[i] * class1_X2[i], class1_X1[i],
                                class1_X2[i], 1])
```

```
 9
10
11 for i in range(len(class2_X1)):
12   trainPoints[i + j, :] = np.array([class2_X1[i] ** 2, class2_X2[i] ** 2,
13                                     class2_X1[i] * class2_X2[i], class2_X1[i],
14                                     class2_X2[i], 1])
15
16 # To check if data is successfully retrieved
17 print(trainPoints)
```

```
[[  4.   4.   4.   2.   2.   1.]
 [  9.   1.   3.   3.   1.   1.]
 [  9.   9.   9.   3.   3.   1.]
 [  1.   9.   3.  -1.  -3.   1.]
 [ 16.   4.   8.   4.   2.   1.]
 [  4.   4.   4.  -2.  -2.   1.]
 [ 16.   9. -12.  -4.   3.   1.]
 [  4.  36.  12.   2.   6.   1.]
 [  0.   0.   0.   0.   0.   1.]
 [  4.   4.  -4.  -2.   2.   1.]
 [  1.   1.   1.  -1.  -1.   1.]
 [ 16.   4.  -8.  -4.   2.   1.]]
```

Normalize class 2

```
1 for i in range(j, 0, -1):
2   trainPoints[-i, :] *= -1
3
4 # To check if data is successfully retrieved
5 print(trainPoints)
```

```
[[  4.   4.   4.   2.   2.   1.]
 [  9.   1.   3.   3.   1.   1.]
 [  9.   9.   9.   3.   3.   1.]
 [  1.   9.   3.  -1.  -3.   1.]
 [ 16.   4.   8.   4.   2.   1.]
 [  4.   4.   4.  -2.  -2.   1.]
 [-16.  -9.  12.   4.  -3.  -1.]
 [ -4. -36. -12.  -2.  -6.  -1.]
 [ -0.  -0.  -0.  -0.  -0.  -1.]
 [ -4.  -4.   4.   2.  -2.  -1.]
 [ -1.  -1.  -1.   1.   1.  -1.]
 [-16.  -4.   8.   4.  -2.  -1.]]
```

Task 3

Use Perceptron Algorithm (both one at a time and many at a time) for finding the weightcoefficients of the discriminant function

```
1 def updateWeight(w, alpha, y_mc, update_cnt):
2   return w + alpha * y_mc, update_cnt + 1 * (1 if sum(y_mc) != 0 else 0)
3
4 def oneTime(y, w, alpha, iteration = 150):
5   it = 0
6   update_cnt = 0
7   points = y.shape[0]
8
9   while it < iteration:
10    res = np.zeros((points, 1))
11    for i in range(points):
12      val = np.dot(y[i],w.T )
13      res[i] = 0 if val > 0 else 1
14      w, update_cnt = updateWeight(w, alpha, res[i] * y[i], update_cnt)
15    it += 1
16    if sum(res) == 0:
17      break
18  return it, update_cnt
19
20 def manyTime(y, w, alpha, iteration = 150):
21   it = 0
22   update_cnt = 0
23   points = y.shape[0]
24
25   while it < iteration:
26     res = np.zeros((points, 1))
27     for i in range(points):
28       val = np.dot(y[i], w.T)
29       res[i] = 0 if val > 0 else 1
30     it += 1
```

```
    if sum(res) == 0:
      break
    else:
      w, update_cnt = updateWeight(w, alpha, sum(res * y), update_cnt)
  return it, update_cnt
```

Task 4

```
alphaAll = [i / 10 for i in range(1, 11, 1)]

np.random.seed(14)
initialweightAll = np.zeros((3, 6))
initialweightAll[1, :] = np.ones((1, 6))
initialweightAll[2, :] = np.random.rand(1, 6)

# To check if data is successfully retrieved
print(alphaAll)
print(initialweightAll)
```

```
[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
[[0.         0.         0.         0.         0.         0.        ]
 [1.         1.         1.         1.         1.         1.        ]
 [0.51394334 0.77316505 0.87042769 0.00804695 0.30973593 0.95760374]]
```

```
output = np.zeros((30, 6))
wType = {0: 'All Zeros', 1: 'All Ones', 2: 'Random'}

for i in range(output.shape[0]):
  alpha = alphaAll[i % 10]
  weight = initialweightAll[i // 10, :].copy()
  itOne, updateOne = oneTime(trainPoints, weight, alpha)
  itMany, updateMany = manyTime(trainPoints, weight, alpha)
  output[i, :] = alpha, i // 10, itOne, updateOne, itMany, updateMany
np.set_printoptions(suppress=True)
print(output)
```

```
[[  0.1   0.   56.  103.  150.  150. ]
 [  0.2   0.   56.  103.  150.  150. ]
 [  0.3   0.   56.  102.  150.  150. ]
 [  0.4   0.   56.  103.  150.  150. ]
 [  0.5   0.   56.  103.  150.  150. ]
 [  0.6   0.   56.  102.  150.  150. ]
 [  0.7   0.   57.  103.  150.  150. ]
 [  0.8   0.   56.  103.  150.  150. ]
 [  0.9   0.   56.  103.  150.  150. ]
 [  1.    0.   56.  103.  150.  150. ]
 [  0.1   1.   50.  101.  150.  150. ]
 [  0.2   1.   54.  102.  150.  150. ]
 [  0.3   1.   38.   76.  150.  150. ]
 [  0.4   1.   57.  102.  150.  150. ]
 [  0.5   1.   56.  102.  150.  150. ]
 [  0.6   1.   55.  101.  150.  150. ]
 [  0.7   1.   56.  102.  150.  150. ]
 [  0.8   1.   55.  102.  150.  150. ]
 [  0.9   1.   63.  115.  150.  150. ]
 [  1.    1.   63.  115.  150.  150. ]
 [  0.1   2.   55.  108.  150.  150. ]
 [  0.2   2.   57.  107.  150.  150. ]
 [  0.3   2.   57.  104.  150.  150. ]
 [  0.4   2.   62.  115.  150.  150. ]
 [  0.5   2.   55.  102.  150.  150. ]
 [  0.6   2.   55.  102.  150.  150. ]
 [  0.7   2.   56.  102.  150.  150. ]
 [  0.8   2.   56.  102.  150.  150. ]
 [  0.9   2.   56.  102.  150.  150. ]
 [  1.    2.   56.  101.  150.  150. ]]
```

Display Plots

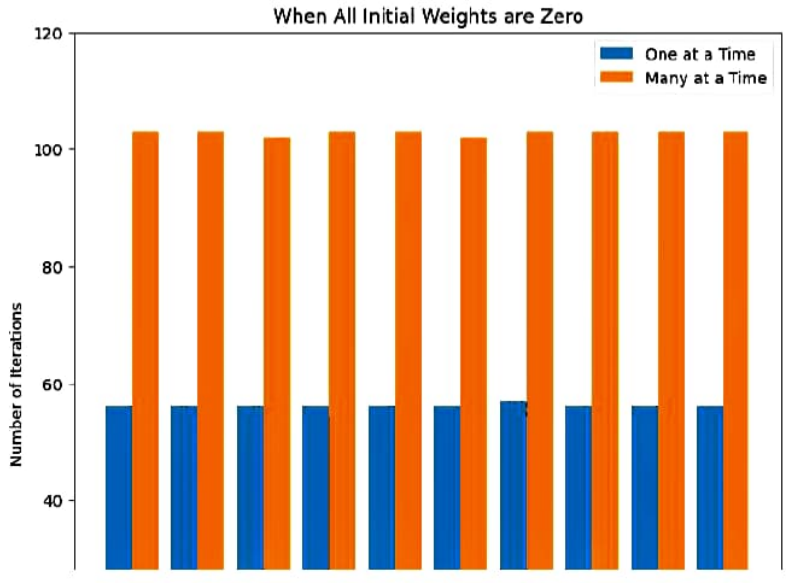
```
print('Found Values')
fig, ax = plt.subplots()
table_data = []

for i in range(output.shape[0]):
  table_data.append([output[i, 0], wType[output[i, 1]], int(output[i, 2]), int(output[i, 4])])

colHeader = ['Learning Rate', 'Initial Weight', 'One at a Time', 'Many at a Time']

table = ax.table(cellText = table_data, colLabels = colHeader, cellLoc = 'center', loc = 'center')

```

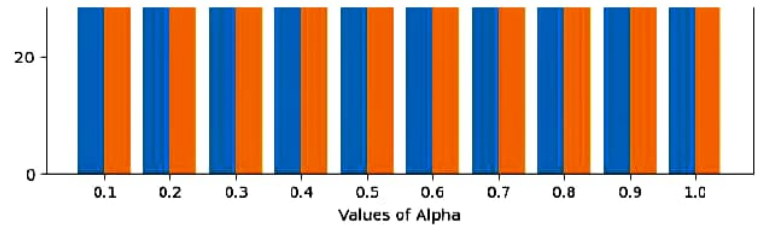
```
table.set_fontsize(25)
table.scale(2,2)
ax.axis('off')
plt.show()

print('\nCharts')
alphaDist = np.arange(len(alphaAll))

plt.figure(figsize = (8, 8))
plt.bar(alphaDist - 0.2, output[0:10, 2], 0.4, label = 'One at a Time')
plt.bar(alphaDist + 0.2, output[0:10, 3], 0.4, label = 'Many at a Time')
plt.xticks(alphaDist, alphaAll)
plt.yticks([i for i in range(0, 121, 20)])
plt.xlabel("Values of Alpha")
plt.ylabel("Number of Iterations")
plt.title("When All Initial Weights are Zero")
plt.legend(loc = 'best')

plt.show()

plt.figure(figsize = (8, 8))
plt.bar(alphaDist - 0.2, output[10:20, 2], 0.4, label = 'One at a Time')
plt.bar(alphaDist + 0.2, output[10:20, 3], 0.4, label = 'Many at a Time')
plt.xticks(alphaDist, alphaAll)
plt.yticks([i for i in range(0, 121, 20)])
plt.xlabel("Values of Alpha")
plt.ylabel("Number of Iterations")
plt.title("When All Initial Weights are One")
plt.legend(loc = 'best')

plt.show()

plt.figure(figsize = (8, 8))
plt.bar(alphaDist - 0.2, output[20:30, 2], 0.4, label = 'One at a Time')
plt.bar(alphaDist + 0.2, output[20:30, 3], 0.4, label = 'Many at a Time')
plt.xticks(alphaDist, alphaAll)
plt.yticks([i for i in range(0, 121, 20)])
plt.xlabel("Values of Alpha")
plt.ylabel("Number of Iterations")
plt.title("When All Initial Weights are Random")
plt.legend(loc = 'best')

plt.show()
```

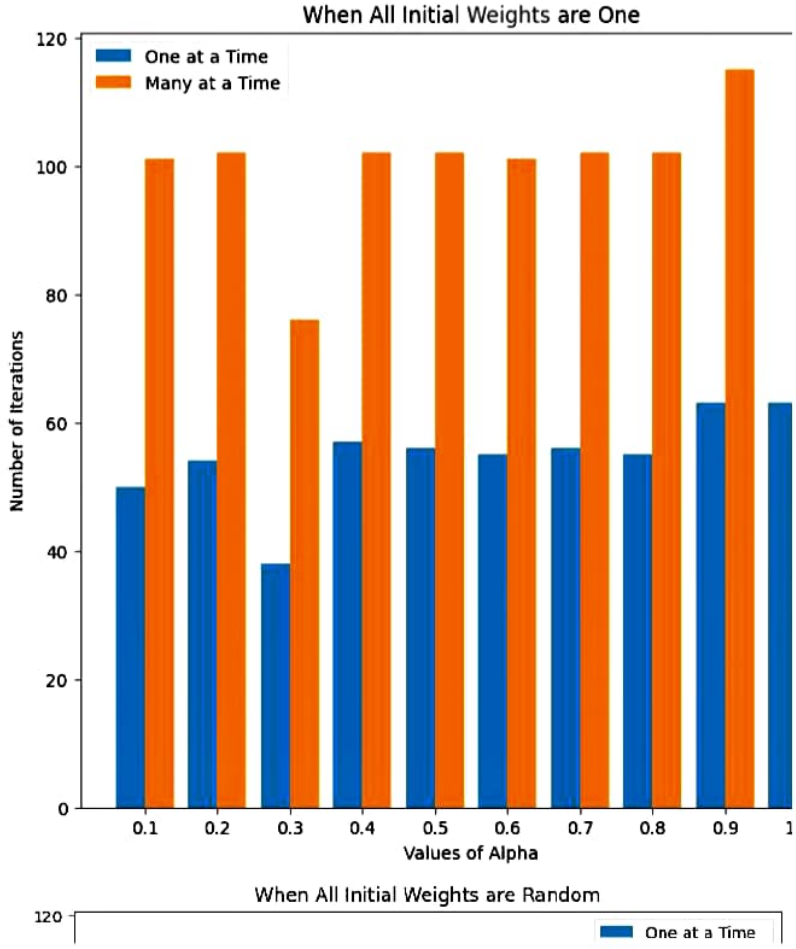
Found Values

Learning Rate	Initial Weight	One at
0.1	All Zeros	5
0.2	All Zeros	5
0.3	All Zeros	5
0.4	All Zeros	5
0.5	All Zeros	5
0.6	All Zeros	5
0.7	All Zeros	5
0.8	All Zeros	5
0.9	All Zeros	5
1.0	All Zeros	5
0.1	All Ones	5
0.2	All Ones	5
0.3	All Ones	3
0.4	All Ones	5
0.5	All Ones	5
0.6	All Ones	5
0.7	All Ones	5
0.8	All Ones	5
0.9	All Ones	6
1.0	All Ones	6
0.1	Random	5
0.2	Random	5
0.3	Random	5
0.4	Random	6
0.5	Random	5
0.6	Random	5
0.7	Random	5
0.8	Random	5
0.9	Random	5
1.0	Random	5

Charts







Question Answer

a. In Task 2, why do we need to take the sample points to a higher dimension?

In Task 2, we need to take the sample points to a higher dimension because the original two-dimensional feature space does not allow for a linear separation of the two classes. When we plotted the training points, we observed that the two classes were scattered in a way that makes it difficult to draw a straight line to separate them effectively. Therefore, we employ a higher-dimensional transformation using the phi (Φ) function.

By transforming the samples to a higher-dimensional space, we aim to find a discriminant function that can effectively separate the classes. In this case, we utilize a second-order polynomial discriminant function, which allows for more complex decision boundaries. This transformation enables the algorithm to capture nonlinear relationships between the features and potentially find a linear boundary in the higher-dimensional space that can separate the classes successfully.

In summary, by taking the sample points to a higher dimension using the phi (Φ) function, we increase the expressive power of our classifier, allowing for the possibility of finding a linear boundary that can separate the classes in a more effective manner.

b. In each of the three initial weight cases and for each learning rate, how many updates does the algorithm take before converging?

```
1 fig, ax = plt.subplots()
2 table_data = []
3
4 for i in range(output.shape[0]):
5     table_data.append([output[i, 0], wType[output[i, 1]], int(output[i, 3]), int(output[i, 5])])
6
```

```
colHeader = ['Learning Rate', 'Initial Weight', 'One at a Time', 'Many at a Time']

table = ax.table(cellText=table_data, colLabels=colHeader, cellLoc='center', loc='center')

table.set_fontsize(12)
table.scale(1.5, 1.5)
ax.axis('off')
plt.show()

```

Learning Rate	Initial Weight	One at a Time	Many at a Time
0.1	All Zeros	103	150
0.2	All Zeros	103	150
0.3	All Zeros	102	150
0.4	All Zeros	103	150
0.5	All Zeros	103	150
0.6	All Zeros	102	150
0.7	All Zeros	103	150
0.8	All Zeros	103	150
0.9	All Zeros	103	150
1.0	All Zeros	103	150
0.1	All Ones	101	150
0.2	All Ones	102	150
0.3	All Ones	76	150
0.4	All Ones	102	150
0.5	All Ones	102	150
0.6	All Ones	101	150
0.7	All Ones	102	150
0.8	All Ones	102	150
0.9	All Ones	115	150
1.0	All Ones	115	150
0.1	Random	108	150
0.2	Random	107	150
0.3	Random	104	150
0.4	Random	115	150
0.5	Random	102	150
0.6	Random	102	150
0.7	Random	102	150
0.8	Random	102	150
0.9	Random	102	150
1.0	Random	101	150

```
1
```