Experiment 3

NAME	**Shreya Shetty**
UID	**2019140059**
CLASS	**TE IT**
BATCH	**B**
ACADEMIC YEAR	**2021-22**
SUBJECT	**SC (Soft Computing)**
COURSE CODE	**IT312**
EXPERIMENT NO.	**3**

Aim:
To implement basic Neural Network rules.

Problem Statement:
Problem To Distinguish Between Apples And Oranges:
A produce dealer has a warehouse that store a variety of fruits & vegetables. When fruit is brought to the warehouse , a various types of fruits may be mixed together. The dealer wants a machine that will sort the fruit according to type . There is a conveyer belt on which the fruit is loaded .This conveyer passes through a set of sensors, which measure three properties of fruits :shape , texture and weight.
Bias= < Any Value>

Type of sensor	Output of sensor	Condition
Shape sensor	1	if fruit is approx. round
	0	if fruit is elliptical.
Texture Sensor	1	If surface is smooth
	0	If surface is rough
Fruit sensor	1	Apple
	0	Orange

Write a program to design a perceptron to recognize these patterns.(Use any Open source tools)
Build a classifier that distinguishes whether a given input is an 'Apple' or 'Orange'. There are two main features that we are going to use, [Shape, Texture], given these two inputs,
our model will predict if it is an Orange=0 or an Apple=1.

Theory:

Perceptron Model

- The Perceptron networks come under single-layer feed-forward networks and are also called simple perceptron's.
- Various types of perceptron's were designed by Rosenblatt (1962) and Minsky-Papert (1969, 1988). However, a simple perceptron network was discovered by Block in 1962.
- The perceptron network consists of three units, namely, the sensory unit (input unit), associator unit (hidden unit), and response unit (output unit).
- The sensory units are connected to associator units with fixed weights having values 1,0 or -1,which are assigned randomly.
- The binary activation function is used in the sensory unit and associator unit.
- The response unit has an activation of 1, 0 or -1. The binary step with a fixed threshold 9 is used as activation for the associator. The output signals that are sent from the associator unit to the response unit are only binary.
- The output of the perceptron network is given by y_out = f(y_{in}) where y_out is activation function and is defined as

$$y\_out = \begin{cases} 1, & if\ \ y\_in > \theta \\ 0, & if\ -\theta \le y\_in \le \theta \\ -1, & if\ \ y\_in < -\theta \end{cases}$$

- The perceptron learning rule is used in the weight updation between the associant unit and the response unit. For each training input, the net will calculate the response and it will determine whether or not an error has occurred.
- The error calculation is based on the comparison of the values with those of the calculated outputs.
- The weights on the connections from the units that send the non zero signal will get adjusted suitably. The weights will be adjusted on the basis of the learning rule if an error has occurred for a particular training pattern
- If no error occurs, there is no weight update and hence the training process may be stopped.
- The below figure shows a simple classification perceptron network :

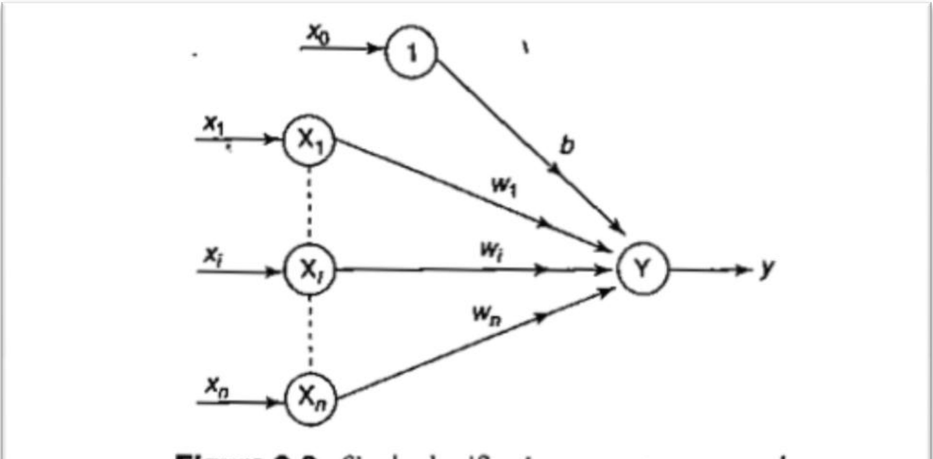


Figure 3-2 Single classification perceptron network.

The notable points regarding Procedure Perceptron-Learning are given below:

1. The input vectors are allowed to be either binary or bipolar. However, the outputs must be in bipolar form.
2. The bias w_0 is adjustable but the threshold q used in the activation function is fixed.
3. Learning takes place only when the computed output does not match the target output. Moreover, as $\Delta w_i = n \times t \times x_i$ the weight adjustment is 0 if $x_i = 0$. Hence, no learning takes place if either the input is 0, or the computed output matches the target output. Consequently, as training proceeds, more and more training patterns yield correct results and less learning is required by the net.
4. The threshold q of the activation function may be interpreted as a separating band of width 2q between the region of positive response and negative response. The band itself is 'undecided' in the sense that if the net input falls within the range [−q, q], the activation is neither positive, nor negative. Moreover, changing the value of q would change the width of the undecided region, along with the position of the separating lines. Therefore, for Perceptrons, the bias and the threshold are no longer interchangeable.
5. The band separating the regions of positive response from that of the negative response is defined by the pair of lines

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = \Theta$$

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = -\Theta$$

Procedure Perceptron-Learning:

Step 1. Initialize all weights, w0, …., wm.

Step 2. Set learning rate h such that $0 < h \leq 1$, and threshold q.

Step 3. For each training pair s : t do Steps 4–8.

Step 4. Activate the input units, xi = si, for i = 0, …., m.

Step 5. Compute the net input to the output unit

$$y\_in = \sum_{i=0}^{m} w_i x_i$$

Step 6. Compute the activation of the output unit using the function

$$y\_out = \begin{cases} 1, & if\ y\_in > \theta \\ 0, & if\ -\theta \le y\_in \le \theta \\ -1, & if\ y\_in < -\theta \end{cases}$$

Step 7. If there is an error, i.e., y_out ≠ t, then adjust the weights as follows

$$w_i\ (new) = w_i\ (old) + \eta \times t \times x_i$$

Step 8. If there were no error, i.e., y_out = t, for the entire set of training pairs, then stop. Otherwise go to Step 3.

Procedure:

1. Import all the required libraries including scikit-learn which is used for importing decision tree to create the reuired classifier.
2. The 2 main features considered are Shape and Texture. Now, train the given data.
3. After that predict the outcome. If the output obtained is ‘0’, result is ‘Apple’, if it is ‘1’, result is ‘Orange’.

Tool/Language:

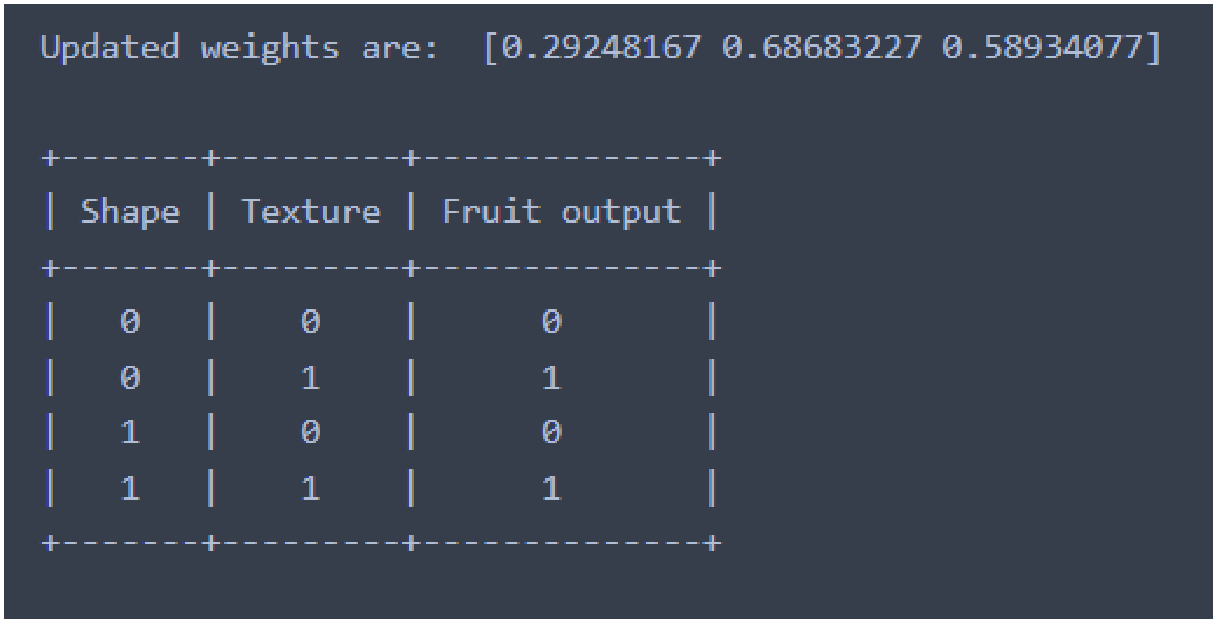
Programming language: Python
IDE USED: Jupyter Notebook
Libraries Used: Random, numpy, prettytable

Code:

```
# Import required libraries
import random
import numpy as np
from prettytable import PrettyTable

# Given Data :
# Shape Sensor - 1=Round, 0=Elliptical
# Texture Sensor - 1=Smooth, 0=Rough
# Fruit Sensor - 1=Apple, 0=Orange

# Activation function
def activation(x,theta):
    if x >= theta:
        return 1
    else:
        return 0

# Train the network
def train_network(inputs,weights,labels,theta):
    lr = 0.4
    n = 30000
    while(n):
        for i in range(labels.shape[0]):
            netInput = weights[0] * inputs[i][0] + weights[1] *
inputs[i][1] + weights[2] # Calculate net input
            output = activation(netInput,theta) # Calculate output
using activation function
            error = (labels[i] - output) # Find error
            weights = weights + lr * error # Update weights
        n = n - 1
    return weights

# Test the network using weights, inputs, activation function
def test_network(inputs,weights,theta):
    netInput = weights[0] * inputs[0] + weights[1] * inputs[1] +
weights[2]
```

```
    output = activation(netInput,theta)
    return output

# Define the inputs
inputs = [[0,0],[0,1],[1,0],[1,1]]
labels = [0,1,0,1]

# Generate the weights randomly
weights = []
x = len(inputs[0])+1
for j in range(x):
    weights.append(random.random())
weights_up =
train_network(np.asarray(inputs),np.asarray(weights),np.asarray(labels
),1)

print("Updated weights are: ",weights_up,"\n") # Print the updated
weights

# Test the network using updated weights
a = test_network([0,0],weights_up,1)
b = test_network([0,1],weights_up,1)
c = test_network([1,0],weights_up,1)
d = test_network([1,1],weights_up,1)

# Print the final result
final_result = PrettyTable(["Shape","Texture","Fruit output"])
final_result.add_row([0,0,a])
final_result.add_row([0,1,b])
final_result.add_row([1,0,c])
final_result.add_row([1,1,d])
print(final_result)
```

Output:



```
Updated weights are:  [0.29248167 0.68683227 0.58934077]


+-------+---------+--------------+
| Shape | Texture | Fruit output |
+-------+---------+--------------+
|   0   |    0    |      0       |
|   0   |    1    |      1       |
|   1   |    0    |      0       |
|   1   |    1    |      1       |
+-------+---------+--------------+
```

Conclusion:

In this experiment, I have successfully built a classifier that distinguishes whether a given input is an ‘Apple’ or ‘Orange’. [Shape, Texture] are the 2 given main features I have used, given these two inputs, the model will predict if it is an Orange=0 or an Apple=1. I have successfully written a program to design a perceptron to recognize these patterns.