SCT Experiment No: 1

Name: Abhishek S Waghchaure

PRN: 1032221714

Dept: FY M Tech DSA(2022-24)

Aim:

Implementing Perceptron with appropriate number of inputs and outputs.

Introduction:

Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.

How Does Perceptron Work?

Perceptron is considered a single-layer neural link with four main parameters. The perceptron model begins with multiplying all input values and their weights, then adds these values to create the weighted sum. Further, this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the step function and is represented by 'f.'

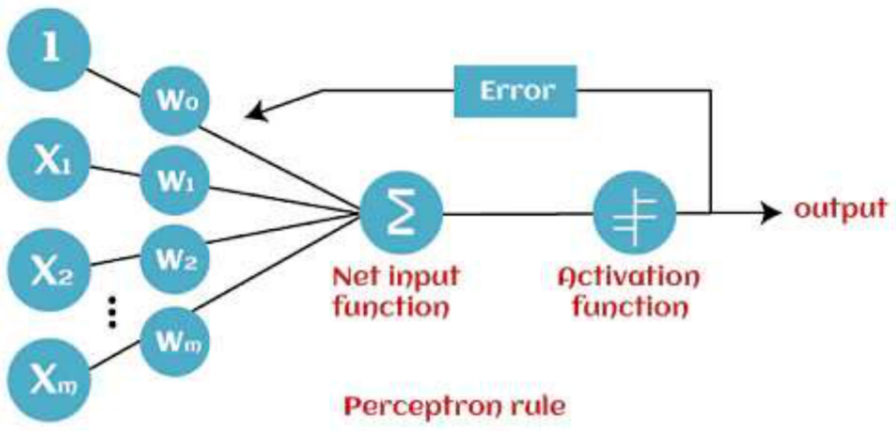
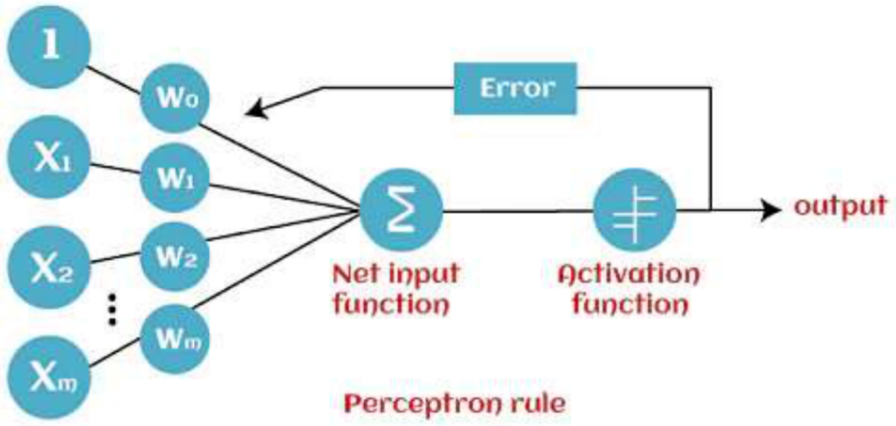


IMAGE COURTESY: javapoint

This step function or Activation function is vital in ensuring that output is mapped between (0,1) or (-1,1). Take note that the weight of input indicates a node's strength. Similarly, an input value gives the ability the shift the activation function curve up or down.



Perceptron rule

OR

NOR

NOT

XOR

XNOR

Code:

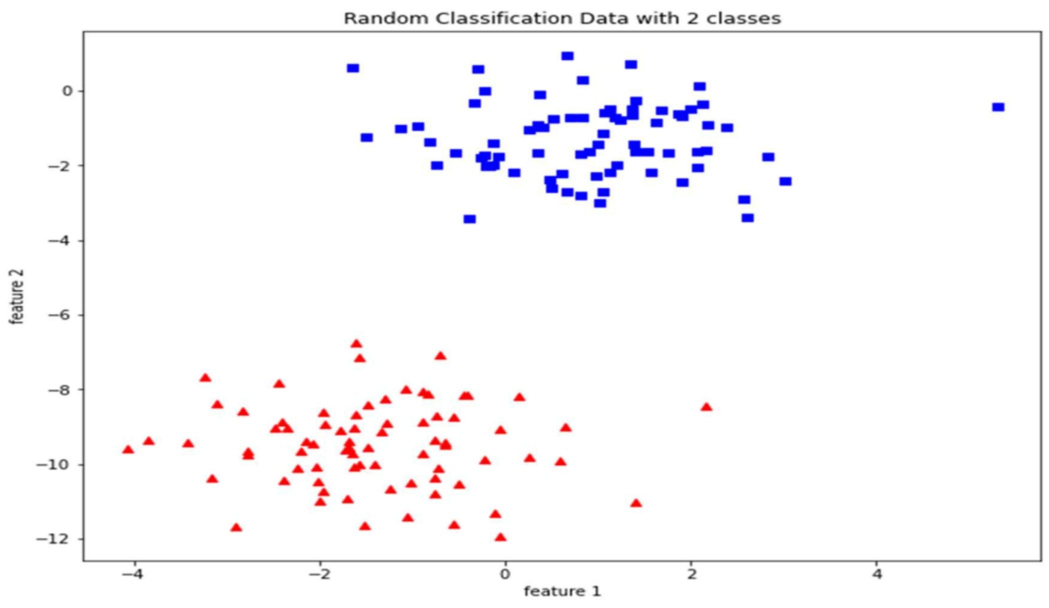
Creating Database for binary classification¶

In [13]:

```
#Creating Database for binary classification

from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
X, y = datasets.make_blobs(n_samples=150,n_features=2,
                           centers=2,cluster_std=1.05,
                           random_state=2)
#Plotting
fig = plt.figure(figsize=(10,8))
plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], 'r^')
plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'bs')
plt.xlabel("feature 1")
plt.ylabel("feature 2")
plt.title('Random Classification Data with 2 classes')
```

Out[13]:

```
Text(0.5, 1.0, 'Random Classification Data with 2 classes')
```



Step function to classify binary¶

In [7]:

```
# Step Function

def step_func(z):
            return 1.0 if (z > 0) else 0.0
```

Perceptron Update Rule¶

In [8]:

```
def perceptron(X, y, lr, epochs):

      #  X --> Inputs.
      #  y --> labels/target.
      #  lr --> learning rate.
      #  epochs --> Number of iterations.

      #  m-> number of training examples
      #  n-> number of features
      m, n = X.shape

      #  Initializing parapeters(theta) to zeros.
      #  +1 in n+1 for the bias term.
      theta = np.zeros((n+1,1))

      #  Empty list to store how many examples were
      #  misclassified at every iteration.
      n_miss_list = []

      # Training.
      for epoch in range(epochs):

            #  variable to store #misclassified.
            n_miss = 0

            #  looping for every example.
            for idx, x_i in enumerate(X):

                  # Insering 1 for bias, X0 = 1.
                  x_i = np.insert(x_i, 0, 1).reshape(-1,1)

                  # Calculating prediction/hypothesis.
                  y_hat = step_func(np.dot(x_i.T, theta))

                  #  Updating if the example is misclassified. if
                  (np.squeeze(y_hat) - y[idx]) != 0:
                        theta += lr*((y[idx] - y_hat)*x_i)

                        #  Incrementing by 1.
                        n_miss += 1

            #  Appending number of misclassified examples
            # at every iteration.
            n_miss_list.append(n_miss)

    return theta, n_miss_list
```

Plotting Decision Boundary¶

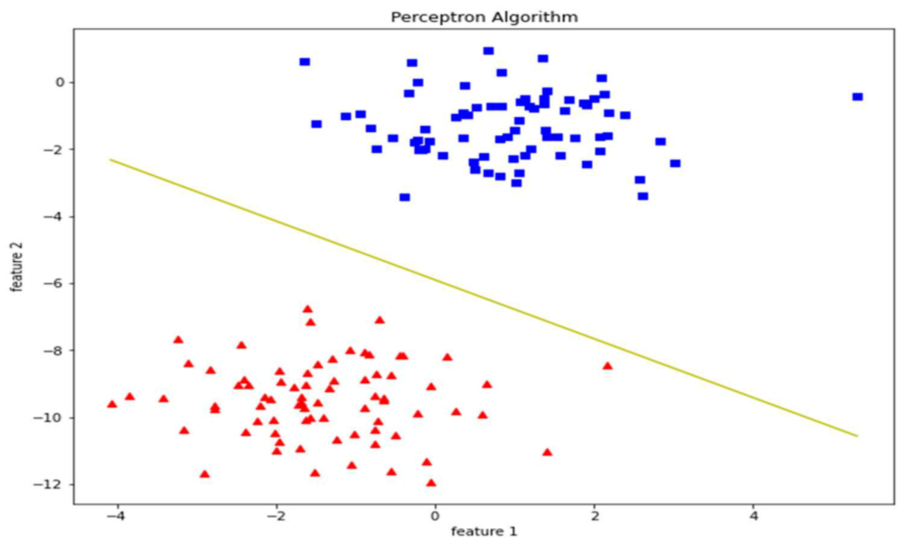
In [11]:

```
def plot_decision_boundary(X, theta):

    #  X --> Inputs
    #  theta --> parameters

    #  The Line is y=mx+c
    #  So, Equate mx+c = theta0.X0 + theta1.X1 + theta2.X2
    #  Solving we find m and c
    x1 = [min(X[:,0]), max(X[:,0])]
    m = -theta[1]/theta[2]
    c = -theta[0]/theta[2]
    x2 = m*x1 + c

    # Plotting
    fig = plt.figure(figsize=(10,8))
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "r^")
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs")
    plt.xlabel("feature 1")
    plt.ylabel("feature 2")
    plt.title('Perceptron Algorithm')
    plt.plot(x1, x2, 'y-')
```

Training and Plotting¶

In [14]:

```
theta, miss_l = perceptron(X, y, 0.5, 100)
plot_decision_boundary(X, theta)
```



AND Gate using Perceptron¶

In [15]:

```
import numpy as np
```

In [23]:

```
#Define unit Step Function
#np.dot() do vector multiplication
def UnitStep(v):
        if v>=0:
                return 1
        else:
                return 0

#design perceptron Model
def perceptron(x,w,b):
        v = np.dot(w,x) + b
        y = UnitStep(v)
        return y
```

In [24]:

```
#  AND Logic Function
#  For AND logic function w1=1, w2=1, b=-1.5 def
AND(x):
        w = np.array([1,1]) b
        = -1.5
        return perceptron(x,w,b)
```

In [27]:

```
#  Testing perceptron Model
test1 = np.array([0,1]) test2 =
np.array([1,1]) test3 =
np.array([0,0]) test4 =
np.array([1,0])
```

In [28]:

```
print("AND({}, {}) = {}".format(0,1,AND(test1)))
print("AND({}, {}) = {}".format(1,1,AND(test2)))
print("AND({}, {}) = {}".format(0,0,AND(test3)))
print("AND({}, {}) = {}".format(1,0,AND(test4)))
```

```
AND(0, 1) = 0
AND(1, 1) = 1
AND(0, 0) = 0
AND(1, 0) = 0
```

OR Gate using Perceptron¶

In [30]:

```
# OR Logic Gate
# For OR logic gate w1=1, w2=1, b=-0.5 def
OR(x):
    w = np.array([1,1]) b
    = -0.5
    return perceptron(x,w,b)
```

In [31]:

```
# Testing perceptron Model
test1 = np.array([0,1]) test2 =
np.array([1,1]) test3 =
np.array([0,0]) test4 =
np.array([1,0])
print("OR({}, {}) = {}".format(0,1,OR(test1)))
print("OR({}, {}) = {}".format(1,1,OR(test2)))
print("OR({}, {}) = {}".format(0,0,OR(test3)))
print("OR({}, {}) = {}".format(1,0,OR(test4)))
```

```
OR(0, 1) = 1
OR(1, 1) = 1
OR(0, 0) = 0
OR(1, 0) = 1
```

NOT Gate using Perceptron¶

In [44]:

```
# NOT Logic Gate
# For NOT logic gate w=-1 b=0.5
def NOT(x):
    w = -1 b
    = 0.5
    return perceptron(x,w,b)


# Testing perceptron Model
test1 = np.array([0]) test2 =
np.array([1])
print("NOT({}) = {}".format(0,NOT(test1)))
print("NOT({}) = {}".format(1,NOT(test2)))
```

```
NOT(0) = 1
NOT(1) = 0
```

NOR Gate using Perceptron¶

In [45]:

```
# NOR Logic Gate
# For NOR logic gate w1=-1, w2=-1, b=0.5 def
NOR(x):
    w = np.array([-1,-1])
```

```
        b = 0.5
        return perceptron(x,w,b)

# Testing perceptron Model
test1 = np.array([0,1]) test2 =
np.array([1,1]) test3 =
np.array([0,0]) test4 =
np.array([1,0])
print("NOR({}, {}) = {}".format(0,1,NOR(test1)))
print("NOR({}, {}) = {}".format(1,1,NOR(test2)))
print("NOR({}, {}) = {}".format(0,0,NOR(test3)))
print("NOR({}, {}) = {}".format(1,0,NOR(test4)))
```

```
NOR(0, 1) = 0
NOR(1, 1) = 0
NOR(0, 0) = 1
NOR(1, 0) = 0
```

NAND Gate using Perceptron¶

In [47]:

```
# NAND Logic Gate
# For NAND logic gate w1=-1, w2=-1, b=1
def NAND(x):
        w = np.array([-1,-1]) b =
        1
        return perceptron(x,w,b)

# Testing perceptron Model
test1 = np.array([0,1])
test2 = np.array([1,1])
test3 = np.array([0,0])
test4 = np.array([1,0])
print("NAND({}, {}) = {}".format(0,1,NAND(test1)))
print("NAND({}, {}) = {}".format(1,1,NAND(test2)))
print("NAND({}, {}) = {}".format(0,0,NAND(test3)))
print("NAND({}, {}) = {}".format(1,0,NAND(test4)))
```

```
NAND(0, 1) = 1
NAND(1, 1) = 0
NAND(0, 0) = 1
NAND(1, 0) = 1
```

XOR using logic¶

In [53]:

```
# XOR Logic
Gate def XOR(x):
        a = AND(x)
        b = OR(x)
```

```
    c = NOT(a)
    d = np.array([c,b])
    output = AND(d)
    #output = AND(NOT(AND(x)),OR(x))
    return output

# Testing perceptron Model
test1 = np.array([0,1]) test2 =
np.array([1,1]) test3 =
np.array([0,0]) test4 =
np.array([1,0])
print("XOR({}, {}) = {}".format(0,1,XOR(test1)))
print("XOR({}, {}) = {}".format(1,1,XOR(test2)))
print("XOR({}, {}) = {}".format(0,0,XOR(test3)))
print("XOR({}, {}) = {}".format(1,0,XOR(test4)))
```

```
XOR(0, 1) = 1
XOR(1, 1) = 0
XOR(0, 0) = 0
XOR(1, 0) = 1
```

XNOR using logic¶

In [57]:

```
# XNOR Logic Gate

def XNOR(x):
    a = OR(x)
    b = AND(x)
    c = NOT(a)
    d = np.array([c,b])
    output = OR(d)
    #output = OR(NOT(OR(x)),AND(x))
    return output

# Testing perceptron Model
test1 = np.array([0,1]) test2 =
np.array([1,1]) test3 =
np.array([0,0]) test4 =
np.array([1,0])
print("XNOR({}, {}) = {}".format(0,1,XNOR(test1)))
print("XNOR({}, {}) = {}".format(1,1,XNOR(test2)))
print("XNOR({}, {}) = {}".format(0,0,XNOR(test3)))
print("XNOR({}, {}) = {}".format(1,0,XNOR(test4)))
```

```
XNOR(0, 1) = 0
XNOR(1, 1) = 1
XNOR(0, 0) = 1
XNOR(1, 0) = 0
```

Conclusion:

In the above process(code), we learned about the Perceptron models, the simplest type of artificial neural network that carries input and their weights, the sum of all weighted information, and an activation function. All the Perceptron models are continuously contributing to AIML. Perceptron models help the computer to work more efficiently on complex problems using Machine Learning technologies. These are the basics of artificial neural networks.

Reference:

https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron

https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron

https://www.geeksforgeeks.org/implementation-of-perceptron-algorithm-for-nand-logic-gate-with-2-bit-binary-input/