ML LAB ASSIGNEMENT - 1

Perceptron – AND, OR and Sample Dataset

NAME : S. S. V. ADITYA NITTALA

ROLL NO: 177163

Salient Features:

- As given in the pdf forwarded by the PhD scholars, the following algorithm was implemented:

Algorithm 1 Perceptron algorithm

```
procedure PERCEPTRON
    for each node x_i ∈ Data do
        if w_t^T x_i > 0 then
            Predict positive label
        else
            Predict negative label
        end if
        if wrong label then
            if true label is positive then
                w_{t+1} = w_t + x_i
            else
                w_{t+1} = w_t - x_i
            end if
        end if
    end for
end procedure
```

- The ***threshold*** function is also same as given in the pdf forwarded.
- The sample dataset is normalized using the z-score normalization.
- Data is split into 90% training and 10% testing.
- For the logic gates, the outputs printed are the bias and the weights.
- For the sample data, the test accuracy is printed.
- Code is attached below.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
X = [0, 0, 0, 1, 1, 0, 1, 1]
X = np.reshape(X, (4, 2))
y_and = [-1, -1, -1, 1]
y_and = np.reshape(y_and, (4, 1))
y_or = [-1, 1, 1, 1]
y_or = np.reshape(y_or, (4, 1))

print(X)
print(X.shape)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
(4, 2)
```

```
def threshold(Z):
    for i in range(Z.shape[0]):
        if Z[i] > 0:
            Z[i] = 1
        else:
            Z[i] = -1
    return Z
```

```
def normalization(X):
    mean = np.mean(X, axis=0)
    std = np.std(X, axis=0)
    X = (X - mean)/std
    return X
```

```
'''Forward and Backward pass implementation'''

def train(X, y, w, b, iters):
    for i in range(iters):
        Z = np.dot(X, w.T) + b
        Z = threshold(Z)
        E = Z - y
        for j in range(E.shape[0]):
            if E[j] != 0:
                if y[j] > 0:
                    w = w + X[j]
                else:
                    w = w - X[j]
```

```
    return w, b


def find_accuracy(X, Y, w, b):
    Z = np.dot(X, w.T) + b
    Z = threshold(Z)
    return accuracy_score(y_test, Z)


w = np.zeros(2)
w = np.expand_dims(w, axis=0)
b = -1

'''AND gate implementation'''
print("AND gate implementation : ")
w, b = train(X, y_and, w, b, 6)
print("Bias : {}".format(b))
print("Weights : \n{}\n".format(w))

w = np.zeros(2)
w = np.expand_dims(w, axis=0)

''' OR gate implementation '''
print("OR gate implementation : ")
w, b = train(X, y_or, w, b, 6)
print("Bias : {}".format(b))
print("Weights : \n{}".format(w))
```

```
AND gate implementation :
Bias : -1
Weights :
[[1. 1.]]

OR gate implementation :
Bias : -1
Weights :
[[2. 2.]]
```

```
''' Perceptron Training Algorithm with sample data '''

Data = pd.read_csv('/percep_data.csv')
Y = Data['Y']
X = Data[['X1', 'X2']]
X = normalization(X)
Y = np.expand_dims(Y, axis=1)
print("Shape of Y : {}".format(Y.shape))
print("Shape of X : {}".format(X.shape))
```

```
Shape of Y : (1000, 1)
Shape of X : (1000, 2)
```

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1, random_state = 42)
```

```
w = np.random.randn(1, 2)
b = np.random.randn()
print("Shape of weights : {}".format(w.shape))
print("Shape of x_train : {}, y_train : {}".format(x_train.shape, y_train.shape))
print(x_train)
```

```
Shape of weights : (1, 2)
Shape of x_train : (900, 2), y_train : (900, 1)
           X1        X2
716  1.698851  0.625282
351 -1.409669  1.409476
936  1.057688 -1.133713
256  0.745459  1.545916
635  1.230740  0.650181
..        ...       ...
106  0.882481  0.818460
270 -1.442591  0.311410
860 -0.963468 -0.862202
435 -1.417628 -0.112863
102  0.697162  1.217073

[900 rows x 2 columns]
```

```
w, b = train(np.array(x_train), np.array(y_train), w, b, 100)
accuracy = find_accuracy(np.array(x_test), np.array(y_test), w, b)
print("Accuracy of model : {}".format(accuracy))
```

```
Accuracy of model : 0.85
```