LAB REPORT

Course No: CSE 4204.

Course title: Sessional based on 4203.

Submitted to:
Dr. Md. Rabiul Islam
Professor,
Department of Computer Science and Engineering,
Rajshahi University of Engineering & Technology.

Submitted by:
Arun Kundu
Roll: 143045
Section: A
4th Year, even Semester.
Department of Computer Science and Engineering.
Rajshahi University of Engineering & Technology.

Name of the problem: Applying single layer perception algorithm on a data set for understanding the time complexity and efficiency.

Introduction: In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. It is a type of linear classifier, i.e. a classification algorithm that makes all of its predictions based on a linear predictor function combining a set of weights with the feature vector. The linear classifier says that the training data should be classified into corresponding categories such that if we are applying classification for two categories, then all the training data must be lie in these two categories.

The binary classifier defines that there should be only two categories for classification.

The basic perceptron algorithm is used for binary classification and all the training examples should lie in these categories. The term comes from the basic unit in a neuron, which is called the perceptron.

Basic Algorithm :

```
        P←inputs with level 1;
        N←inputs with level 0;
        Initialized w randomly
        While !convergence do
        Pick random x € P U N ;
        If  x € P  and w.x ≤  0 then
                W = w +  x;
        If  x € N  and w.x ≥ 0;
                W = w -  x;
        End
End
```

Code :

```
import random
import numpy as np
import time
threshold = 0.5
n_feature = 0
learningRate = 0.001
weights = []
def train1(trainData, trainClass):
    global n_feature, weights
    n_feature = len(trainData[0])
    weights = [random.random() for _ in range(n_feature)]
    index = 0
    while(index < len(trainData)):
        data = trainData[index]
        summation = 0
        i = 0
        while(i < n_feature):
            summation += data[i] * weights[i]
            i += 1
        classLabel = -1
        if summation < threshold:
            classLabel = 0
        else:
            classLabel = 1
        if trainClass[index] == classLabel:
            index += 1
        else:
            value = trainClass[index] -
classLabel
            i = 0
            while(i < n_feature):
                weights[i] = weights[i] + value *
learningRate * data[i]
                i += 1
            index = 0

def train2(trainData, trainClass):
    global n_feature, weights
    n_feature = len(trainData[0])
    weights = [random.random() for _ in
range(n_feature)]

    trainData = np.array(trainData)
    trainClass = np.array(trainClass)
    trainDataA =
trainData[np.where(trainClass == 0)[0]]
    trainClassA =
trainClass[np.where(trainClass == 0)[0]]
    trainDataB =
trainData[np.where(trainClass == 1)[0]]
    trainClassB =
trainClass[np.where(trainClass == 1)[0]]

    index = 0
    while(index < len(trainData)):
        while index < len(trainDataA):
            data = trainDataA[index]
            summation = sum(data * weights)
            if summation < threshold:
                classLabel = 0
            else:
                classLabel = 1
            if trainClassA[index] == classLabel:
                index += 1
            else:
                value = trainClassA[index] -
classLabel
                weights += value * data *
learningRate
                index = 0

        indexB = 0
        changed = False
        while indexB < len(trainDataB):
            data = trainDataB[indexB]
            summation = sum(data * weights)
            if summation < threshold:
                classLabel = 0
            else:
                classLabel = 1
            if trainClassB[indexB] ==
classLabel:
                indexB += 1
            else:
                value = trainClassB[indexB] -
classLabel
                weights += value * data *
learningRate
                indexB = 0
                changed = True

        if changed:
            index = 0
        else:
            break

def test(testData):
    predict = []
    global n_feature, weights
    idx = 0
    while(idx < len(testData)):
        i = 0
        summation = 0
        while(i < n_feature):
            summation += weights[i] *
testData[idx][i]
            i += 1
        if summation < threshold:
            predict.append(0)
        else:
            predict.append(1)
        idx += 1
    return predict

number = 128
data = []
for i in range(number):
    binary = bin(i)[2:]
    binary = (len(bin(number)[2:]) - 
len(binary) - 1) * '0' + binary
    binaryList = []
    for digit in list(binary):
        digit = int(digit)
        binaryList.append(digit)
    data.append(binaryList)

className = [0 for i in 
range(int(number/2))] + [1 for i in 
range(int(number/2))]


from sklearn.model_selection import 
train_test_split
from sklearn.metrics import accuracy_score

for i in range(1, 6):
    train_data, test_data, train_class, 
test_class = train_test_split(data, className, 
test_size=i/10)

    print('Train: ', (10 - i)*10, '% Test:', i*10, 
'%')
    print('-------First method------------')
    startTime = time.time()
    train1(train_data, train_class)
    endTime = time.time()
    predictedClass = test(test_data)

    print('Accuracy: ', 
accuracy_score(test_class, predictedClass))
    print('Time: ', round(endTime - startTime, 
2))


    print('-------Second method------------')
    startTime = time.time()
    train2(train_data, train_class)
    endTime = time.time()
    predictedClass = test(test_data)

    print('Accuracy: ', 
accuracy_score(test_class, predictedClass))
    print('Time: ', round(endTime - startTime, 
2))

print('==========================
============')
```

Result :

Train data	Test data	First method		Second method	
		Accuracy	Time	Accuracy	Time
90%	10%	1.0	0.03	1.0	0.1
80%	20%	0.96153846538	0.6	0.884615384615	0.09
70%	30%	0.974358974359	0.03	0.948717948718	0.03
60%	40%	1.0	0.05	0.980769230769	0.03
50%	50%	0.875	0.02	0.875	0.05
90%	10%	1.0	0.07	0.923076923077	0.08
80%	20%	1.0	0.05	0.884615384615	0.06
70%	30%	0.974358974359	0.0	0.897435897436	0.09
60%	40%	0.980769230769	0.02	0.961538461538	0.06
50%	50%	0.984375	0.05	0.96875	0.06
90%	10%	1.0	0.06	1.0	0.03
80%	20%	1.0	0.02	1.0	0.06
70%	30%	0.974358974359	0.06	0.871794871795	0.22
60%	40%	0.923076923077	0.03	0.942307692308	0.13
50%	50%	0.984375	0.03	0.96875	0.08

Conclusion :

From the result it can be conclude that, First algorithm is better for time complexity and second algorithm is better for accuracy.