Implementing the Perceptron Algorithm for Finding the Weights of a Linear Discriminant Function

Mayisha Farzana
dept.Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204028@aust.edu

***Abstract*—The purpose of this experiment is to apply it to Algorithm of the perceptron to find the weights of a linear discriminant. A Perceptron is an algorithm used for the supervised learning of binary classifiers. Perceptrons play an important role in binary classification. We start with random weights in this algorithm, and gradually, we will forward to the actual weights. There are two implementations of this algorithm: batch processing, which is also known as many at a time, and the other is a single update known as one at a time. We will use some sample data for both processes, and we will compare the performance of these two methods with different learning rates.**

***Index Terms*—Samples, Dimension, Normalization, Second order polynomial, classified, misclassified, learning rate, weight.**

I. Introduction

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers: functions that can decide whether an input (represented by a vector of numbers) belongs to one class or another. To draw the decision boundary line, we need weights. In non linear data, if we use perceptron algorithm, we will not get the correct hyperplane. But if we take it in higher dimension, it can correctly work. In lower dimension, points are not linearly separable. For it, we will use ϕ function to take it in higher dimension. To update the weights, there are two processes: single update and batch update. For updating the weights, we will use the following equations,

$$w(i+1) = w(i) + \alpha y_m^(k) \quad if w^T(i) y_m^(k) <= 0$$

where $y_m^(k)$ is misclassified samples and α is learning rate. and the another is

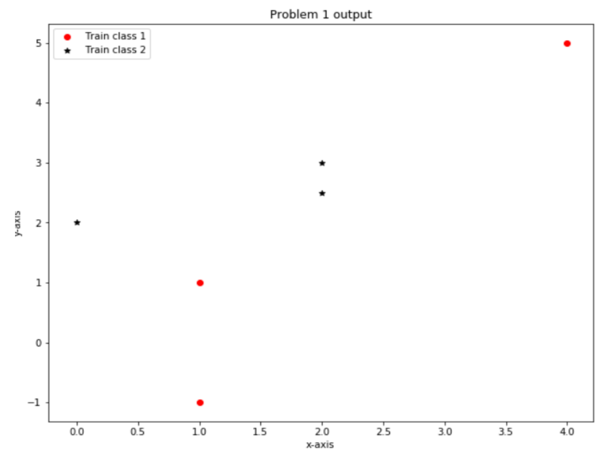
$$w(i) \quad if w^T(i) y_m^(k) > 0$$

which is correctly classified. If the data points are misclassified, we need to update the weight.

II. Experimental Design / Methodology

A. Plotting the sample data points and observing:

At first, we will train the sample dataset. I have used numpy to train the dataset. After plotting the datapoints, for same class I have use same color and marker. To plot the dataset, I have used Matplotlib package.



After plotting the points, we can see that it cannot be separate using one linear decision boundary. Every time there will be some error. So we will have to use a ϕ function to move to a higher dimension.We have used the following functions for it.

$$y = [x_1^2 \quad x_2^2 \quad x_1 * x_2 \quad x_1 \quad x_2 \quad 1]$$

B. Calculating high dimensional sample points:

Now we will take the sample points into a higher dimensional sample points. The given ϕ function moves our sample points to a six dimensional space. Before using gradient descent technique, we have to normalize any one of the class. Normalization or reflection is a process to shift one class completely to the opposite. It can be done only in two class problem. Here we normalize class two. After negating it, we took the class for further computation.

C.Applying perceptron method:

Then I have calculated the weights for both one at a time(single perceptron) and many at a time (batch perceptron) for different learning rate between 0.1 to 1 with step size 0.1.

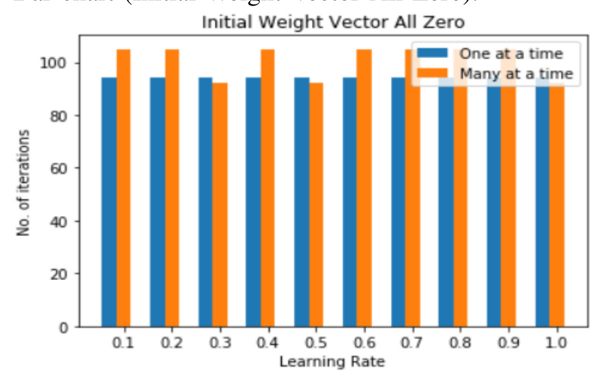
D. Showing the results:

Now after calculating the iterations for both cases we have found three table with weight initiated with all zero's, all one's and randomly. tables and outputs are given below:

Sample Output(Initial Weight Vector All Zero):

Alpha (Learning Rate)	One at a time	Many at a time
0.10	94	105
0.20	94	105
0.30	94	92
0.40	94	105
0.50	94	92
0.60	94	105
0.70	94	105
0.80	94	105
0.90	94	105
1.00	94	92

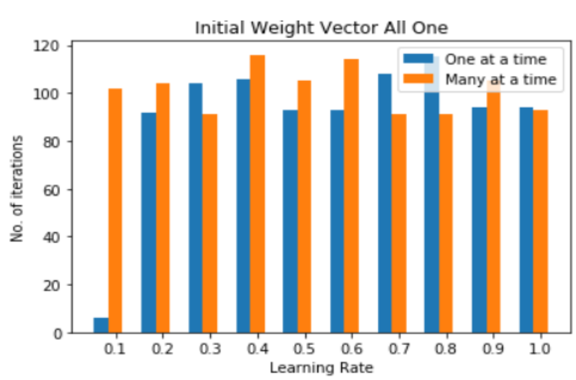
Bar chart (Initial Weight Vector All Zero):



Sample Output(Initial Weight Vector All One):

Alpha (Learning Rate)	One at a time	Many at a time
0.10	6	102
0.20	92	104
0.30	104	91
0.40	106	116
0.50	93	105
0.60	93	114
0.70	108	91
0.80	115	91
0.90	94	105
1.00	94	93

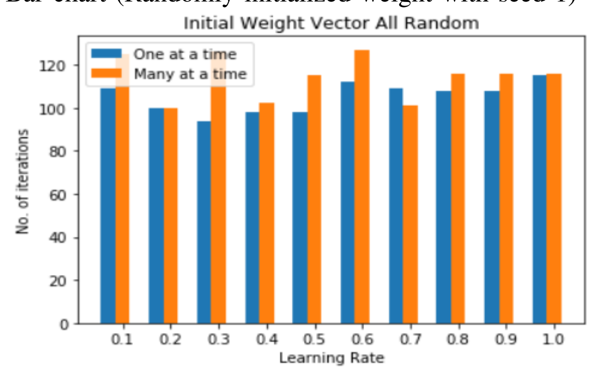
Bar chart (Initial Weight Vector All One):



Sample Output (Randomly initialized weight with seed 1):

Alpha (Learning Rate)	One at a time	Many at a time
0.10	109	125
0.20	100	100
0.30	94	126
0.40	98	102
0.50	98	115
0.60	112	127
0.70	109	101
0.80	108	116
0.90	108	116
1.00	115	116

Bar chart (Randomly initialized weight with seed 1)



III. Question Answering

a. In task 2, we need to take the sample points into a higher dimension because if it is non linear data, we cannot get the correct hyperplane. If the datasets are linear, then the decision boundary can perfectly work. But in non linear data the data's are not linearly separable.Fot it, we need to take it to a higher dimension where it can classify correctly.To take the data points into higher dimension, we have used ϕ function.

b. In each of the three initial weight cases and for each learning rate the number of updates the algorithm take before converging is given in the upper section using three tables. I have also shown the bar chart to visualize the results.

IV. Result Analysis

In this experiment both one at a time and many at a time approach for Perceptron Algorithm was performed. We can clearly see that many at a time takes much more time than one at a time. Here are only 6 datasets. It is possible that if there are much more dataset then one at a time would be much better than many at a time. The main reason of one at a time is better because it updates itself every time, where as many at a time do not.

V. Conclusion

The perceptron algorithm is one of the most commonly used machine learning algorithms for binary classification. Here we have learned to implement perceptron after taking the data to a higher dimension where we can separate the classes using a linear line. Though the solution stops, when it gets all the data's perfectly classified, but the solution space is very large in this case. We conclude from the experiment that one at a time approach of Perceptron algorithm is far better than the many at a time approach because in one at a time values are updated every time in contrast to many at a time.

VI. Algorithm Implementation / Code

```python
'''Problem 1 '''
import numpy as np
import matplotlib.pyplot as plt
ar=np.loadtxt('train-perceptron.txt',dtype='float64')#loading the train and test data
print(ar)
s=len(ar)
print(s)
for x in  ar:
    print("x = ",x[0],"y = ",x[1],"class = ",x[2]) # print the class names
ar_clas1=np.array([row for row in ar if row[2]==1])
print(ar_clas1)
ar_clas2=np.array([row for row in ar if row[2]==2])
print(ar_clas2)
x_train_1=ar_clas1[:,0]
y_train_1=ar_clas1[:,1]
x_train_2=ar_clas2[:,0]
y_train_2=ar_clas2[:,1]
class1_len=len(ar_clas1)
print(class1_len)
class2_len=len(ar_clas2)
print(class2_len)
ar_clas1=np.array([row for row in ar if row[2]==1])
print(ar_clas1)
ar_clas2=np.array([row for row in ar if row[2]==2])
print(ar_clas2)
fig,ax=plt.subplots()#to show it in the same figure
plt.title("Problem 1 output")
plt.xlabel('x-axis', color='black')
plt.ylabel('y-axis', color='black')
ax.scatter(x_train_1,y_train_1,marker='o',color='r',label='Train class 1')
ax.scatter(x_train_2,y_train_2,marker='*',color='black',label='Train class 2')
fig.set_figheight(8)
fig.set_figwidth(10)
ax.legend()#show the output figure

'''Problem 2'''
##########PHI FUNCTION###########
def get_phii(x1,x2):
    return np.array([x1*x1,x2*x2,x1*x2,x1,x2,1])
y=get_phii(2,3)
print(y)
hd_y=[]#taking one list and store the class value
#Storing the class values
for row in ar_clas1:
    hd_y.append(get_phii(row[0],row[1]))
for row in ar_clas2:
     hd_y.append(np.dot(get_phii(row[0],row[1]),-1))
     #Normalization: Negating here all the values to classify it
for x in hd_y:
    print(x)
#############  Batch Processing Function ####
w=np.zeros_like(hd_y[0])
def perceptron_many_at_a_time(learning_rate, w):
    for itr in range(1000000000):
        mc = False;#At first misclassify will be false that means the datas are misclassified
        sum_y = np.zeros_like(hd_y[0])
        for i in range(len(hd_y)):
            val = np.dot(hd_y[i], w)#multiply the y values with weight vector in matrix multiplication
            if (val <= 0.0):#Checking it if it is greater than zero whcih means it is correctly classified
                mc = True#misclassified is true that means we do not need to update it's weight vector
                sum_y = sum_y + hd_y[i]#the values which are misclassified we , adding the y values in here
        sum_y = sum_y * learning_rate# alpha*sum(y)
        w = w + sum_y#w=w+aplha*sum(y)
        if (mc == False):
            return itr + 1#if the datas are misclassified we need to do it again and we will count the iteration number
    return -1 #when there will be no misclassified values we will return -1 fa
#############  Single Processing Function ####
w=np.zeros_like(hd_y[0])
def perceptron_one_at_a_time(learning_rate, w):
    for itr in range(1000000000):
        mc = False;
        for i in range(len(hd_y)):
            val = np.dot(hd_y[i], w)
            if (val <= 0.0):
                mc = True
                sum_y = np.zeros_like(hd_y[0])
                sum_y = sum_y + hd_y[i]
                sum_y = sum_y * learning_rate
                w = w + sum_y
        if (mc == False):
            return itr + 1
    return -1

w=np.zeros_like(hd_y[0])
print("Initial Weight Vector = All Zero")
print("Alpha(Learning Rate)"+"\t\t"+"One at a Time"+"\t\t"+'Many at a Time')
for learning_rate in np.arange(0.1,1.1,0.1):
    print("\t{:.2f}".format(learning_rate)+'\t\t\t\t'+str(perceptron_one_at_a_time(learning_rate,w)) + "\t\t\t" + str(perceptron_many_at_a_time(learning_rate, w)))



w=np.ones_like(hd_y[0])
print("Initial Weight Vector = All One")
print("Alpha(Learning Rate)"+"\t\t"+"One at a Time"+"\t\t"+'Many at a Time')
for learning_rate in np.arange(0.1,1.1,0.1):
    print("\t{:.2f}".format(learning_rate)+'\t\t\t\t'+str(perceptron_one_at_a_time(learning_rate,w)) + "\t\t\t" + str(perceptron_many_at_a_time(learning_rate, w)))


np.random.seed(1)
# weight vector zero
w = np.random.uniform(0, 1, len(hd_y[0]))
print("Initial Weight Vector = All Random")
print("Alpha(Learning Rate)"+"\t\t"+"One at a Time"+"\t\t"+'Many at a Time')
for learning_rate in np.arange(0.1, 1.1, 0.1):
    print("\t{:.2f}".format(learning_rate) + "\t\t\t\t" + str(perceptron_one_at_a_time(learning_rate, w))+ "\t\t\t" + str(perceptron_many_at_a_time(learning_rate, w)))

#############Bar Graph###############
```

```python
##############Weight Vector Zero###########
x_label = []
for x in np.arange(0.1, 1.1, 0.1):
    s = "{:.1f}".format(x)
    x_label.append(s)

one_at_a_time = []
w=np.zeros_like(hd_y[0])
tmp = []
for learning_rate in np.arange(0.1, 1.1, 0.1):
    a = perceptron_one_at_a_time(learning_rate, w)
    one_at_a_time.append(a)
many_at_a_time = []
for learning_rate in np.arange(0.1, 1.1, 0.1):
    b = perceptron_many_at_a_time(learning_rate, w)
    many_at_a_time.append(b)

bar_width = 0.3
index = np.arange(10)
plt.title('Initial Weight Vector All Zero')
plt.bar(index, one_at_a_time, bar_width,label='One at a time')
plt.bar(index + bar_width, many_at_a_time, bar_width, label='Many at a time')
plt.xlabel('Learning Rate')
plt.ylabel('No. of iterations')
plt.xticks(index + bar_width, x_label)
plt.legend()
plt.show()

################Bar Graph############
################Weight Vector One###############
x_label = []
for x in np.arange(0.1, 1.1, 0.1):
    s = "{:.1f}".format(x)
    x_label.append(s)

one_at_a_time = []
w=np.ones_like(hd_y[0])
tmp = []
for learning_rate in np.arange(0.1, 1.1, 0.1):
    a = perceptron_one_at_a_time(learning_rate, w)
    one_at_a_time.append(a)
many_at_a_time = []
for learning_rate in np.arange(0.1, 1.1, 0.1):
    b = perceptron_many_at_a_time(learning_rate, w)
    many_at_a_time.append(b)

bar_width = 0.3
index = np.arange(10)
plt.title('Initial Weight Vector All One')
plt.bar(index, one_at_a_time, bar_width,label='One at a time')
plt.bar(index + bar_width, many_at_a_time, bar_width, label='Many at a time')
plt.xlabel('Learning Rate')
plt.ylabel('No. of iterations')
plt.xticks(index + bar_width, x_label)
plt.legend()
plt.show()
################Bar Graph############
################Weight Vector Random##############

x_label = []
for x in np.arange(0.1, 1.1, 0.1):
    s = "{:.1f}".format(x)
    x_label.append(s)

one_at_a_time = []
np.random.seed(1)
w = np.random.uniform(0, 1, len(hd_y[0]))
tmp = []
for learning_rate in np.arange(0.1, 1.1, 0.1):
    a = perceptron_one_at_a_time(learning_rate, w)
    one_at_a_time.append(a)
many_at_a_time = []
for learning_rate in np.arange(0.1, 1.1, 0.1):
    b = perceptron_many_at_a_time(learning_rate, w)
    many_at_a_time.append(b)

bar_width = 0.3
index = np.arange(10)
plt.title('Initial Weight Vector All Random')
plt.bar(index, one_at_a_time, bar_width,label='One at a time')
plt.bar(index + bar_width, many_at_a_time, bar_width, label='Many at a time')
plt.xlabel('Learning Rate')
plt.ylabel('No. of iterations')
plt.xticks(index + bar_width, x_label)
plt.legend()
plt.show()
```