Implementing the Perceptron algorithm for finding the weights of a Linear Discriminant function.

Md. Zahid Fesabelilla
Computer Science and Engineering. Ahsanullah University of Science and Technology
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204082@aust.edu

***Abstract*—This document is about implementation of the Perceptron algorithm for finding the weights of a Linear Discriminant function.**

***Index Terms*—Perceptron algorithm, Many at a time, One at a time.**

I. Introduction

Linear Discriminant function is a method used in statistics and other fields, to find a linear combination of features that characterizes or separates two or more classes of objects or events.
The Perceptron algorithm helps us to find a weights vector that classifies all points.
Here, we have a set of data that contain two classes. Our task is to find a weight vector using perceptron algorithm which is to classify all points.

II. Experimental Design / Methodology

Firstly, Plot all sample points from both classes,then identify the same class using the same color and marker.
These two classes can not be separated with a linear boundary.
Secondly, Generate the high dimensional sample points y,here

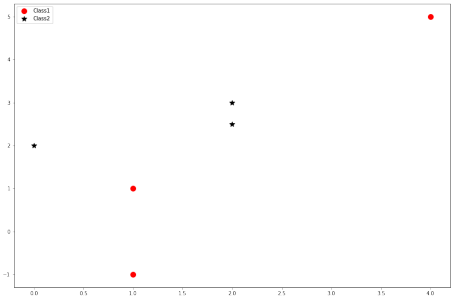

Fig. 1. All sample points from both classes

we using the following the formula for both classes to convert them to six dimensions.

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

Thirdly , The Perceptron Algorithm (both one at a time and many at a time) finding the weight coefficients of the discriminant function (i.e., values of w) boundary for linear classifier in task 2.
Here alpha is the learning rate and alpha is between 0 and 1
Initially , weight = [0 0 0 0 0 0] and learning rate between

$$\underline{w}(i+1) = \underline{w}(i) + \alpha \underline{\tilde{y}}_m^{(k)} \quad \text{if } \underline{w}^T(i)\, \underline{\tilde{y}}_m^{(k)} \le 0$$

$$\text{(i.e., if } \underline{\tilde{y}}_m^{(k)} \text{ is misclassified)}$$

$$= \underline{w}(i) \quad \text{if } \underline{w}^T(i)\, \underline{\tilde{y}}_m^{(k)} > 0$$

0 and 1.
In many at a time W update like a batch so its called batch update. But, in one at a time , at a time one W updated so its called single update.
In many at a time, we multiply Y and W. If any of the values are less than or equal to zero then we will update the weight. Until all values will greater than zero.
For update weight, we take summation of Y and multiply it to learning rate and sum it to the previous weight.
In one at a time, we are doing the same process but here we update weight for single point of Y.

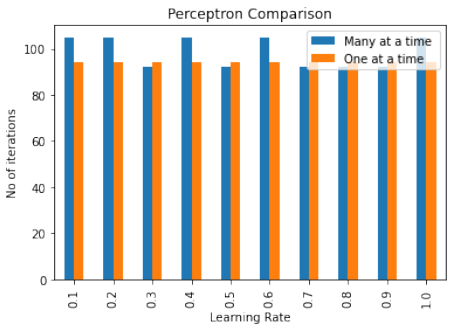

Fig. 2. When initial weights all are zero.

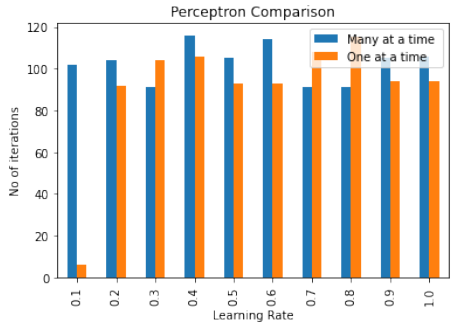

Fig. 3. When initial weights all are one.

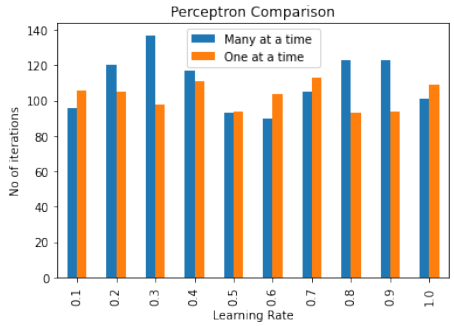

Fig. 4. When initial weights all are random variable.

III. Result Analysis

When we use initially all weights zero then one at a time needs less iteration than many at a time.
Here we take the same learning rate for all combinations.
For random weight, and initial all weight one perform like same.

IV. Conclusion

When we perform Perceptron Algorithm (both one at a time and many at a time) and we find finally the updated weight for high dimensions which classify all high dimensional points.

a. In task 2, why do we need to take the sample points to a high dimension?

Ans : When data are in a lower dimension it is hard to classify linearly. For this reason, we use the FI function to convert it from a lower dimension to a higher dimension. Its helps us to classify data linearly.

b. In each of the three initial weight cases and for each learning rate, how many updates does the algorithm take before converging?
Ans :

	Many at a time	One at a time	Learning Rate
0	105	94	0.1
1	105	94	0.2
2	92	94	0.3
3	105	94	0.4
4	92	94	0.5
5	105	94	0.6
6	92	94	0.7
7	92	94	0.8
8	92	94	0.9
9	105	94	1.0

Fig. 5. For all zero

	Many at a time	One at a time	Learning Rate
0	102	6	0.1
1	104	92	0.2
2	91	104	0.3
3	116	106	0.4
4	105	93	0.5
5	114	93	0.6
6	91	108	0.7
7	91	115	0.8
8	105	94	0.9
9	106	94	1.0

Fig. 6. For all one

	Many at a time	One at a time	Learning Rate
0	96	106	0.1
1	120	105	0.2
2	137	98	0.3
3	117	111	0.4
4	93	94	0.5
5	90	104	0.6
6	105	113	0.7
7	123	93	0.8
8	123	94	0.9
9	101	109	1.0

Fig. 7. For randomly initialized with seed fixed

V. Algorithm Implementation / Code

```
# coding: utf-8

# In[213]:


import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[214]:


import numpy as np
import math
import random


# In[215]:


x1,y1 = [], []
x2,y2 = [], []

for line in open('train-perceptron.txt', 'r'):
    values = [float(s) for s in line.split()]
    if(values[2] == 1):
        x1.append(values[0])
        y1.append(values[1])

    if(values[2] == 2):
        x2.append(values[0])
        y2.append(values[1])


# In[216]:


plt.figure(figsize=(15, 10))

red_dot, = plt.plot(x1,y1,'ro',markersize = 10) #
    red dot
blk_str, = plt.plot(x2,y2,'ok',marker='*',markersize
    =10) # black *

plt.legend([red_dot, blk_str], ["Class1", "Class2"])


plt.plot()
plt.show()


# In[217]:


#for class 1
sixDC1 = []
for i in range(len(x1)):
    l1 = []
    X1 = pow(x1[i],2)
    l1.append(X1)
    X2 =  pow(y1[i],2)
    l1.append(X2)
    X3 = x1[i]*y1[i]
    l1.append(X3)
    l1.append(x1[i])
    l1.append(y1[i])
    l1.append(1)
    sixDC1.append(l1)

print(sixDC1)
```

```
# In[218]:


#for class 2
sixDC2 = []
for i in range(len(x2)):
    l1 = []
    X1 =-1 * pow(x2[i],2)
    l1.append(X1)
    X2 = -1 * pow(y2[i],2)
    l1.append(X2)
    X3 =-1 * x2[i] * y2[i]
    l1.append(X3)
    l1.append(-1 * x2[i])
    l1.append(-1 * y2[i])
    l1.append(-1)

    sixDC2.append(l1)

print(sixDC2)


# In[219]:


sixDClasses = sixDC1 + sixDC2
sixDClasses = np.array(sixDClasses)
print(sixDClasses)


# In[252]:


#w = np.array([0,0,0,0,0,0])
#w = np.array([1,1,1,1,1,1])

#'''
random.seed(10)
A = []
for i in range(int(6)):
     A.append(random.random())

w = np.array(A)
#'''
w


# In[253]:


#answer = [all(n >= 1 for n in i) for i in
    sixDClasses]
#answer
answer = False
data1 = []
data2 = []


# In[254]:


'''
for i in range(len(answer)):
    if(bool(answer[i])==True):
        print(sixDClasses[i])
'''


# In[255]:


#learning rate is 1
def manyAtATime(lst,w,n):
```

```
    flag = 0
    updtW = w
    WY = []
    forW = []
    temp = []
    j = 0
    #print(lst)
    while flag != 1:
        j = j+1

        for i in range(int(6)):
            temp = sum(lst[i] * updtW)
            WY.append(temp)
            #rint("Temp : ",temp)
       # print(WY)
        answer = [all(i>0 for i in WY)]
        #print(answer)
        if(answer[0] == False):
            for i in range(len(updtW)):
                if(WY[i] <= 0):
                    #print(WY[i]," ",lst[i])
                    forW.append(lst[i])
                    temp = np.array(forW)
        else:
            print(j)
            flag = 1
            data1.append(j)
            break
        #print("temp",temp)
        WY = []
        forW= []
        temp1 = (temp.sum(axis = 0))
        updtW = updtW + n * temp1
        #print(j)
        #print("Update Weight : ",updtW)



# In[256]:


a = 10
n = 0.0
for i in range(a):
    n = n + .1
    print("N = ",n,"\n")
    manyAtATime(sixDClasses,w,n)
    print("--------------------END
    ----------------------")
    print("\n")


# In[257]:


#for one at a time
def singleUpdate(lst,w,n):
    flag = 0
    updtW = w
    WY = []
    j = 0
    while flag != 1:
        for i in range(len(updtW)):
            temp = sum(lst[i] * updtW)
            if(temp>0):
                updtW = updtW
            else:
                updtW = updtW + n*lst[i]
                #print("Update Weight : ",updtW)
            WY.append(temp)
            #rint("Temp : ",temp)

        j = j + 1
        print(j)
       #print("WY",WY)
        answer = [all(i>0 for i in WY)]
        print(answer,"\n")
        if(answer[0] == True):
            #print(WY)
            data2.append(j)
            break
        else:
            WY = []



# In[258]:


#singleUpdate(sixDClasses,w,.1)


# In[259]:


a = 10
N = 0.0
for i in range(a):
    N = N + .1
    print("N = ",N,"\n")
    singleUpdate(sixDClasses,w,N)
    print("--------------------END
    ----------------------")
    print("\n")



# In[260]:


data1


# In[261]:


data2


# In[262]:


import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import matplotlib.pyplot as plt


# In[263]:


learningRate = ['0.1','0.2','0.3','0.4','0.5','0.6',
    '0.7','0.8','0.9','1.0']
df = pd.DataFrame({"Many at a time ": data1, "One at
     a time ": data2}, index=learningRate)
x = df.plot.bar()
x.set_title('Perceptron Comparison')
x.set_xlabel('Learning Rate')
x.set_ylabel('No of iterations')


# In[264]:


learningRate =
    [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
table = {"Learning Rate ": learningRate," Many at a
    time ": data1," One at a time ": data2}
```

```
df = pd.DataFrame(table)
df


# In[ ]:
```

REFERENCES