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

Shimul Paul
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204014@aust.edu

***Abstract*—Perceptron is an algorithm used for the supervised learning of binary classifiers.It means that perceptron is a line drawing algorithm between two class where the data points are taken higher dimensions from lower dimensions.In this perceptron algorithm there are two types.One is batch process and another is single process.Here we tried to implement this algorithm for finding the perfect weights of a linear discriminant function.Also here we tried to compare the performance of this two types with various learning rate.**

***Index Terms*—Perceptron,Linear Discriminant Function,Batch Process,Single Process**

I. Introduction

Perceptron is an algorithm that basically used for the supervised learning of binary classifiers.It is used for draw a straight line between two class.For this straight line we need weights.For linear data we cannot get the proper values.So that we need to higher dimension data.So for convert the lower level data in higher level we need ϕ function.There are two process in this weight update process.One is one at a time and another is many at a time.For this update the equations are following bellow :

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

Where $y_m^{(k)}$ is misclassified samples and learning rate is α.And the another is

$$w(i) \quad if w^T(i) y_m^{(k)} > 0 \tag{2}$$

Which is correctly classified.

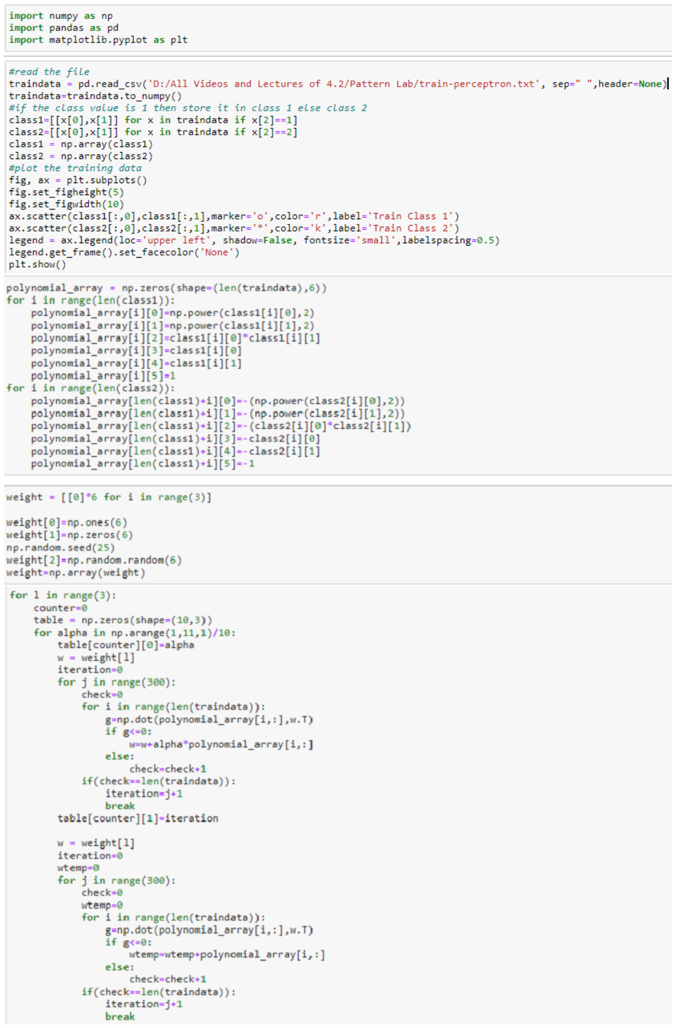
II. Experimental Design / Methodology

- Firstly, I read all the training data using pandas read_csv() function and store it in a array.
- Then, for the training data I store the values of two feature in two arrays if the class is 1 then class1 else in class2.
- After that, I plot the training class1 and training class2 values using matplotlib with marker and color.
- Then, as we see that it can not separable by one decision boundary.So now i implement the higher dimension rule for getting the data in higher dimension.The equation i follow is ;

$$y = [{x_1}^2 \quad {x_2}^2 \quad x_1 * x_2 \quad x_1 \quad x_2 \quad 1] \tag{3}$$

- Then, I have normalized the class2 data points by negating them and took them in one side for further computation.
- Then, I have calculated the weights for both one at a time and many at a time for different learning rate between 0.1 to 1 with a step size 0.1.

III. Algorithm Implementation / Code



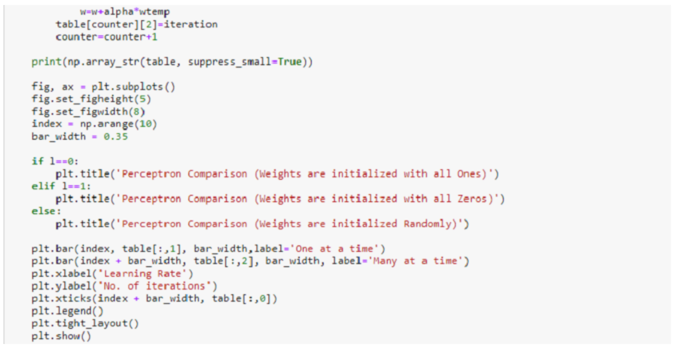
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
#read the file
traindata = pd.read_csv('D:/All Videos and Lectures of 4.2/Pattern Lab/train-perceptron.txt', sep=" ",header=None)
traindata=traindata.to_numpy()
#if the class value is 1 then store it in class 1 else class 2
class1=[[x[0],x[1]] for x in traindata if x[2]==1]
class2=[[x[0],x[1]] for x in traindata if x[2]==2]
class1 = np.array(class1)
class2 = np.array(class2)
#plot the training data
fig, ax = plt.subplots()
fig.set_figheight(5)
fig.set_figwidth(10)
ax.scatter(class1[:,0],class1[:,1],marker='o',color='r',label='Train Class 1')
ax.scatter(class2[:,0],class2[:,1],marker='*',color='k',label='Train Class 2')
legend = ax.legend(loc='upper left', shadow=False, fontsize='small',labelspacing=0.5)
legend.get_frame().set_facecolor('None')
plt.show()
```

```
polynomial_array = np.zeros(shape=(len(traindata),6))
for i in range(len(class1)):
    polynomial_array[i][0]=np.power(class1[i][0],2)
    polynomial_array[i][1]=np.power(class1[i][1],2)
    polynomial_array[i][2]=class1[i][0]*class1[i][1]
    polynomial_array[i][3]=class1[i][0]
    polynomial_array[i][4]=class1[i][1]
    polynomial_array[i][5]=1
for i in range(len(class2)):
    polynomial_array[len(class1)+i][0]=-(np.power(class2[i][0],2))
    polynomial_array[len(class1)+i][1]=-(np.power(class2[i][1],2))
    polynomial_array[len(class1)+i][2]=-(class2[i][0]*class2[i][1])
    polynomial_array[len(class1)+i][3]=-class2[i][0]
    polynomial_array[len(class1)+i][4]=-class2[i][1]
    polynomial_array[len(class1)+i][5]=-1
```

```
weight = [[0]*6 for i in range(3)]

weight[0]=np.ones(6)
weight[1]=np.zeros(6)
np.random.seed(25)
weight[2]=np.random.random(6)
weight=np.array(weight)
```

```
for l in range(3):
    counter=0
    table = np.zeros(shape=(10,3))
    for alpha in np.arange(1,11,1)/10:
        table[counter][0]=alpha
        w = weight[l]
        iteration=0
        for j in range(300):
            check=0
            for i in range(len(traindata)):
                g=np.dot(polynomial_array[i,:],w.T)
                if g<=0:
                    w=w+alpha*polynomial_array[i,:]
                else:
                    check=check+1
            if(check==len(traindata)):
                iteration=j+1
                break
        table[counter][1]=iteration

        w = weight[l]
        iteration=0
        wtemp=0
        for j in range(300):
            check=0
            wtemp=0
            for i in range(len(traindata)):
                g=np.dot(polynomial_array[i,:],w.T)
                if g<=0:
                    wtemp=wtemp+polynomial_array[i,:]
                else:
                    check=check+1
            if(check==len(traindata)):
                iteration=j+1
                break
```

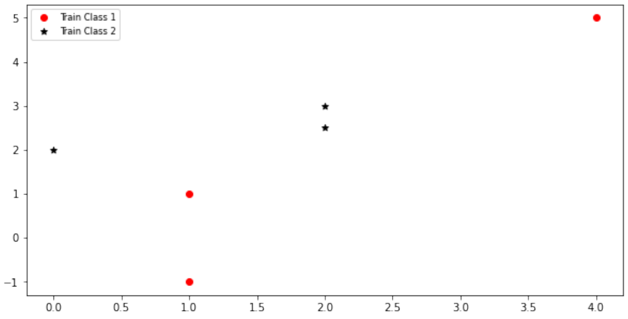


```
            w=w+alpha*wtemp
        table[counter][2]=iteration
        counter=counter+1

print(np.array_str(table, suppress_small=True))

fig, ax = plt.subplots()
fig.set_figheight(5)
fig.set_figwidth(8)
index = np.arange(10)
bar_width = 0.35

if l==0:
    plt.title('Perceptron Comparison (Weights are initialized with all Ones)')
elif l==1:
    plt.title('Perceptron Comparison (Weights are initialized with all Zeros)')
else:
    plt.title('Perceptron Comparison (Weights are initialized Randomly)')

plt.bar(index, table[:,1], bar_width,label='One at a time')
plt.bar(index + bar_width, table[:,2], bar_width, label='Many at a time')
plt.xlabel('Learning Rate')
plt.ylabel('No. of iterations')
plt.xticks(index + bar_width, table[:,0])
plt.legend()
plt.tight_layout()
plt.show()
```

IV. Result Analysis

The results is shown below:

- Firstly, for plotting the training set i got the following output.

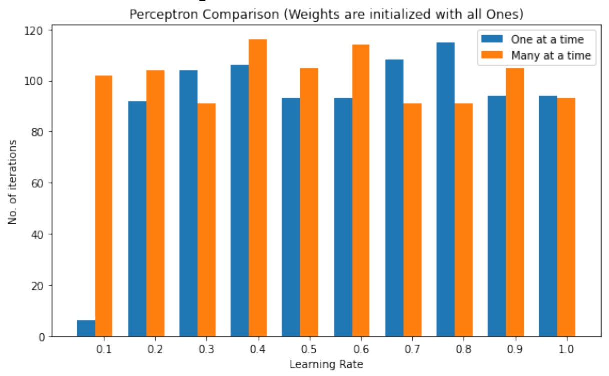


- Secondly, I got perceptron comparison with weights all are initialized with ones.

Perceptron Comparison (Weights are initialized with all Ones)

Learning Rate	One at a time	Many at a time
0.10	6.0	102.0
0.20	92.0	104.0
0.30	104.0	91.0
0.40	106.0	116.0
0.50	93.0	105.0
0.60	93.0	114.0
0.70	108.0	91.0
0.80	115.0	91.0
0.90	94.0	105.0
1.00	94.0	93.0

Barchart (Weights are initialized with all Ones)

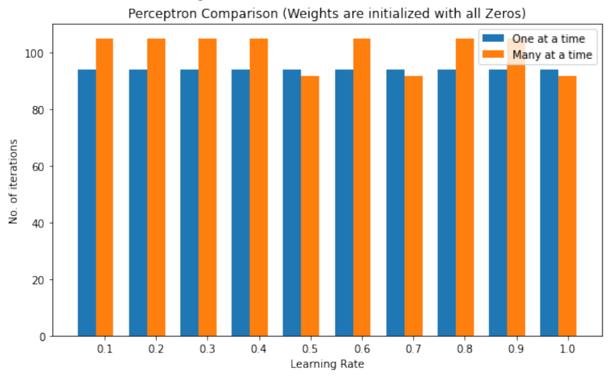


- Thirdly, I got perceptron comparison with weights all are initialized with zeros.

Perceptron Comparison (Weights are initialized with all Zeros)

Learning Rate	One at a time	Many at a time
0.10	94.0	105.0
0.20	94.0	105.0
0.30	94.0	105.0
0.40	94.0	105.0
0.50	94.0	92.0
0.60	94.0	105.0
0.70	94.0	92.0
0.80	94.0	105.0
0.90	94.0	105.0
1.00	94.0	92.0

Barchart (Weights are initialized with all Zeros)

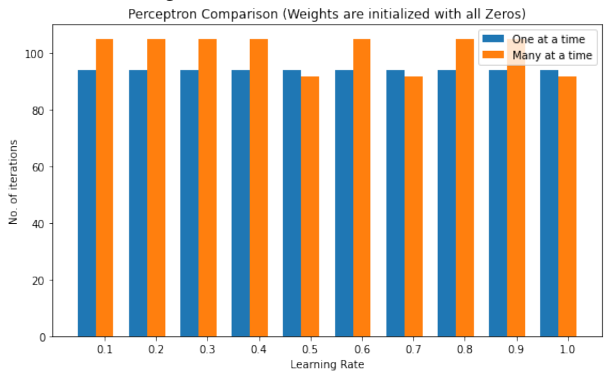


- Finally, I got perceptron comparison with weights all are initialized with random values.

Perceptron Comparison (Weights are initialized Randomly)

Learning Rate	One at a time	Many at a time
0.10	100.0	93.0
0.20	100.0	108.0
0.30	110.0	116.0
0.40	96.0	122.0
0.50	94.0	106.0
0.60	106.0	120.0
0.70	115.0	106.0
0.80	105.0	88.0
0.90	110.0	101.0
1.00	109.0	101.0

Barchart (Weights are initialized with random values)



V. Question Answering

- We need to take the higher dimension sample points because in the lower dimension the points are not linearly separable.For this we take the higher dimension so that we can separate them easily.
- I showed the information of update before converging shown in the upper tables and bar charts.

Conclusion

This perceptron algorithm is a very simple algorithm.It can classify data in a very simple way.Here in this algorithm it uses higher dimension data for classify linearly.This classifier can classify the data in a very small iteration.So, it is a faster classifier.