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

Devopriya Tirtho
16.02.04.033
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh

***Abstract*—In 'Machine Learning', for predicting the accurate class of an unknown sample we need to visualize our training samples whether they are linearly separable or not. 'Perceptron Algorithm' helps to find the actual weights of a 'Linear Discriminant' function for convergence.**

***Index Terms*—Machine Learning, Perceptron Algorithm, Linear Discriminant, Function.**

I. Introduction

'Perceptron Algorithm' is the type of learning algorithm which helps to find the weights to converge a linear discriminant function. In **'Perceptron Learning Algorithm'**, the weights are updated systematically to ommit the misclassified classification. This process leads to converge the linear discriminant function so that, no point is misclassified. This algorithm works on misclassified datapoints by updating weights with a particular learning rate to converge. The learning rate may vary.

II. Task

There is a dataset named 'train-perceptroptron' where all the datapoints are given. The datapoints are classified between **'two'** classes namely $'1'$ and $'2'$. Here are the datapoints of 'train-perceptron' dataset classified into $Class-1$ as $W1$ and $Class-2$ as $W2$:
$W1$= $\{(1,1),(1,-1),(4,5)\}$
$W2$= $\{(2,2.5),(0,2),(2,3)\}$

- Firstly, we have to plot all the data points of each class with corresponding marker in graph.
- Secondly, we have to apply the following phi function in order to get the high dimensional sample points y. The phi function is:

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

- Thirdly, we have to normalize $class\ 2$ and then, we have to use perceptron algorithm on the dataset to find all the coefficients' weights of the discriminant function for the linear classifier. We have to find the weights in two methods. Such as: $One\ at\ a\ Time$ and $Many\ at\ a\ Time$ and have to vary the value of learning rate $alpha$ among 0 to 1 with step size 0.1. The perceptron algorithm which we follow:

$$\tilde{W}(i+1) = \tilde{W}(i) + \alpha\tilde{Y}_m^k \tag{2}$$

if $\tilde{W}(i) * \alpha\tilde{Y}_m^k <= 0; Misclassified$

$$\tilde{W}(i+1) = \tilde{W}(i) \tag{3}$$

if $\tilde{W}(i) * \alpha\tilde{Y}_m^k > 0; Classified$

- Finally, we have to design the perceptron algorithm for initial weights of all zero, all one and randomly initialized with seed fixed and plot all the data points in a bar chart to visualize our experiment.

III. Experimental Design

- **Plotting All Training Points:** In our given dataset which is named as 'train', we need to pick the values which represent X-coordinate values and Y-coordinate values. Then with appropriate marker and distinct color we have to plot the points of **'Class 1'** and **'Class 2'**.

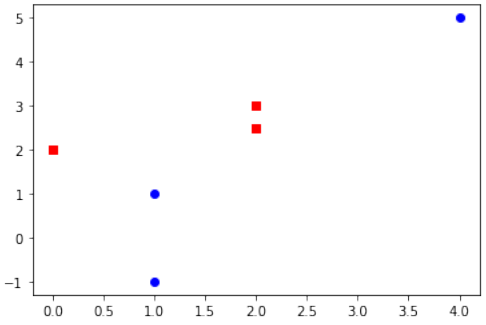

Fig. 1. Datapoints of Class 1 and Class 2

- **Running Perceptron Algorithm:** Then we have to run the perceptron algorithm for all initial weights of zero, one and randomly initialized with fixed seed respectively.
- **Plotting Bar Chart for Visualization:** After getting our desired table, we plot the data into a bar char for visualization.

TABLE I
TABLE FOR INITIAL WEIGHT VECTOR OF ALL ONE

Alpha (Learning Rate)	*One at a Time*	*Many at a Time*
0.1	**6**	**102**
0.2	**92**	**104**
0.3	**104**	**91**
0.4	**106**	**116**
0.5	**93**	**105**
0.6	**93**	**114**
0.7	**108**	**91**
0.8	**115**	**91**
0.9	**94**	**105**
1.0	**94**	**93**

TABLE II
TABLE FOR INITIAL WEIGHT VECTOR OF ALL ZERO

Alpha (Learning Rate)	*One at a Time*	*Many at a Time*
0.1	**94**	**105**
0.2	**94**	**105**
0.3	**94**	**92**
0.4	**94**	**105**
0.5	**94**	**92**
0.6	**94**	**92**
0.7	**94**	**92**
0.8	**94**	**105**
0.9	**94**	**105**
1.0	**94**	**92**

TABLE III
TABLE FOR INITIAL WEIGHT VECTOR OF ALL RANDOMLY INITIALIZED WITH FIXED SEED 3

Alpha (Learning Rate)	*One at a Time*	*Many at a Time*
0.1	**33**	**48**
0.2	**21**	**93**
0.3	**89**	**107**
0.4	**10**	**13**
0.5	**80**	**10**
0.6	**8**	**81**
0.7	**100**	**89**
0.8	**93**	**132**
0.9	**97**	**7**
1.0	**98**	**99**

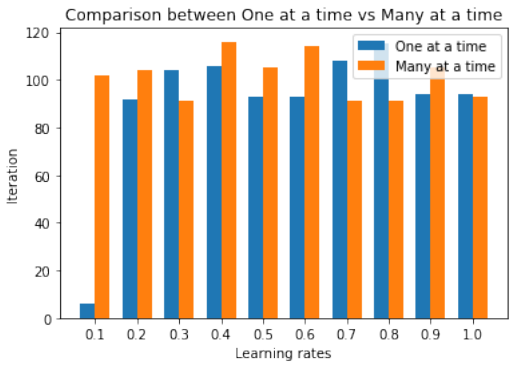

Fig. 2. Bar Chart with Initial Weight Vector of all One

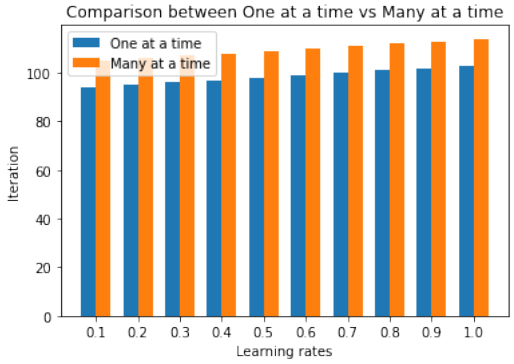

Fig. 3. Bar Chart with Initial Weight Vector of all Zero

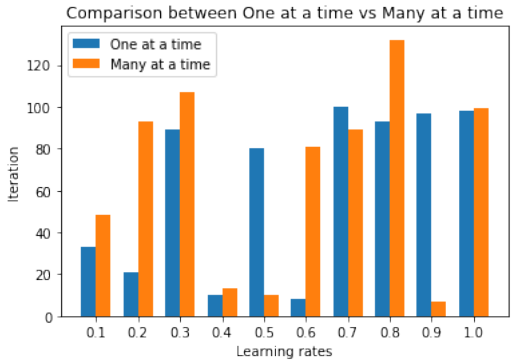

Fig. 4. Bar Chart with Initial Weight Vector of all Randomly Initialized Value with Seed 3

IV. ANSWER OF THE QUESTIONS

Answers:

a. We need to take the sample points to a high dimension to make the points separable. If we do not take the sample points to a higher dimension, they cannot be separated by a single linear line. That is why we need to take them to a high dimension.

b. In each of the three initial weight cases and for each learning rate, the number of updates is required is shown in the above table which is needed by the algorithm take before converging

V. RESULT ANALYSIS

From the implemetation of the **Perceptron Algorithm** it is quite visible that when we vary the learning rate the convergence rate of $single\ update$ and $batch\ update$ varies. When all the initial weights are One with the increment of learning rate $batch\ update$ needs fewer iterations for convergence. $batch\ update$ converges faster than $single\ update$ $four$ times when all the weight vectors are initialized with one. When the weight vector is initialized with $zero$, for all learning rate $single\ update$ needs $ninety-four$ iterations to converge and $batch\ update$ needs higher number of interations for convergence. For random initialization, $single\ update$ converges $seven$ times faster than $batch\ update$.

VI. Python Code

```
[4] train_mat=train_data.to_numpy()
    w_class_1=[]
    w_class_2=[]
    w_class_1_x=[]
    w_class_1_y=[]
    w_class_2_x=[]
    w_class_2_y=[]
    for i in range(6):
      if train_data.at[i,'class']==1:
        w_class_1.append(train_data.at[i,'x1'])
        w_class_1.append(train_data.at[i,'y1'])
        w_class_1_x.append(train_data.at[i,'x1'])
        w_class_1_y.append(train_data.at[i,'y1'])
      else:
        w_class_2.append(train_data.at[i,'x1'])
        w_class_2.append(train_data.at[i,'y1'])
        w_class_2_x.append(train_data.at[i,'x1'])
        w_class_2_y.append(train_data.at[i,'y1'])
```

```
[5] w_class_1

    [1, 1.0, 1, -1.0, 4, 5.0]
```

```
[6] w_class_2

    [2, 2.5, 0, 2.0, 2, 3.0]
```

```
[7] plt.scatter(w_class_1_x,w_class_1_y, color='blue', marker='o')
    plt.scatter(w_class_2_x,w_class_2_y, color='red', marker='s')
```

```
#For initial weight of all 1
learning_rate = np.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])

report = []


for alpha in learning_rate:
    OneIteration = 0
    weight = np.float64(np.array([1, 1, 1, 1, 1, 1]))


    while True:
        OneIteration = OneIteration+1
        MissClassifiedOne = False

        for i in range(0, 6):
            g = np.dot(weight, input[i])

            if(g<=0):
                weight = weight + np.dot(input[i], alpha)
                MissClassifiedOne = True

        if(MissClassifiedOne == False):
            break


    ManyIteration=0
    weight = np.float64(np.array([1, 1, 1, 1, 1, 1]))
    ManyIteration=0
    weight = np.float64(np.array([1, 1, 1, 1, 1, 1]))

    while True:

        temp=np.float64(np.array([0 , 0 , 0 , 0 , 0 , 0]))
        ManyIteration=ManyIteration+1
        MissClassifiedMany=False
        for i in range (0,6):

            g=np.dot(weight,input[i])


            if (g<=0):
                temp+=np.dot(input[i],alpha)
                MissClassifiedMany=True
        weight=weight+temp


        if (MissClassifiedMany==False):
            break

    report.append([alpha,OneIteration, ManyIteration])

report=np.array(report)
final = pd.DataFrame({'alpha': report[:, 0], 'One at a time': report[:, 1],
                      'Many at a time': report[:, 2]})
final
```

Fig. 5. Snapshot of python code

VII. Conclusion

'Perceptron Algorithm' is a simple classification technique which uses linear discriminant function to update the misclassified weights to converge the function accurately. This algorithm works on the misclassified datapoints and helps to find the actual weight vector for the classification model.