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

Ahmad Subaktagin Jabir
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204061@aust.edu

***Abstract*—The main goal of this experiment is to implement the perceptron algorithm for finding the weights of linear discriminant function. In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. There are two types of approaches for the Perceptron Algorithm. They are - One at a Time and Many at a Time. Some sample datas will be used for both approaches and we will compare them with different learning rates and different initial weights.**

***Index Terms*—Perceptron, Sample, Dimension, Hyperplane, Linear, Normalization, Learning Rate, Classified, Misclassified, Weight.**

I. Introduction

A Perceptron is an algorithm used for supervised learning of binary classifiers. Binary classifiers decide whether an input, usually represented by a series of vectors, belongs to a specific class. We need weights to draw a decision boundary. If we use perceptron algorithm in non linear data, we will not get the correct hyperplane. It can correctly work if we take it in a higher dimension as points are linearly separable here. Points are not linearly separable in lower dimension. We will use the ϕ function to take it in a higher dimension. There are two processes to update the weights. They are- One at a time or Single Update and Many at a time or Batch Update. We will use following formula to update the weights:

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

Here $\widetilde{\underline{\mathrm{y}}}_m^{(k)}$ is misclassified samples and α is the learning rate. On the other hand,

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

Which is correctly classified. We need to update the weights if data points are misclassified.

II. Experimental Design / Methodology

A. **Plotting the sample data points and observing:** There are two classes given in the dataset. I used the same color and same marker to distinguish one class from another. Two numpy arrays are used here to store the data from the dataset and plot them with different markers.

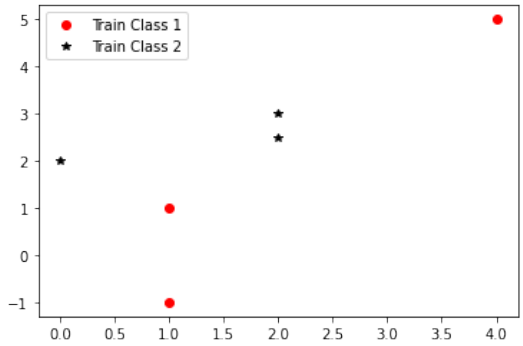

Fig. 1. Plotting The Sample Data Points

After plotting, we can see that one linear decision boundary can not separate it. There will always be some errors. So we have to use a ϕ function to move to a higher dimension.

B. **Calculation of High Dimensional sample points:** We will use following functions to calculate ϕ function:

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

Now, we will use our sample points in the given function. We also have to normalize any one of the classes before using the gradient descent technique. Normalization is mainly a process that shifts one class to its opposite. This process can only be done in two class problems. Here I used $class\ 2$ for normalization. After normalizing, we used the normalized class for further calculations.

C. **Using Perceptron Method:** Then I have calculated weights for both one at a time and many at a time. I have used following equation to calculate One at a time perceptron

$$w(t+1) = w(t) + \eta y$$

And following equation for Many at a time perceptron

$$w(t+1) = w(t) + \eta \sum_{\forall y\ misclassified} y$$

In this step I have calculated with the help of initial weights consisting of all ones and learning rate with 0.1.

D. **Using Perceptron method with different initial weights and different learning rates:** Here I used three different initial weights for both One at a time and Many at a time perceptron. They are- All One, All Zero, Randomly initialized with a fixed seed value which is 10. Also, I used different learning rates between 0.1 and 1 with step size 0.1. I used different tables for three different initial weights containing learning rate, number of iterations for Single and Batch perceptron. I also created three different bar charts visualizing my table data.

III. Result Analysis

After completing all tasks, results are given below:

1) **Calculation of High dimensional sample points:** After calculation I get following values:

```
array([[ 1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  1.  ],
       [ 1.  ,  1.  , -1.  ,  1.  , -1.  ,  1.  ],
       [16.  , 25.  , 20.  ,  4.  ,  5.  ,  1.  ],
       [-4.  , -6.25, -5.  , -2.  , -2.5 , -1.  ],
       [-0.  , -4.  , -0.  , -0.  , -2.  , -1.  ],
       [-4.  , -9.  , -6.  , -2.  , -3.  , -1.  ]])
```

Fig. 2. High Dimensional Sample Points

2) **Using Perceptron Method:** With using all one initial weights and 0.1 learning rate, I got 6 iterations for Single Perceptron and 102 for Batch Perceptron.

3) **Using the Perceptron method with different initial weights and different learning rates:** With using different initial weights and different learning rates I get various values for both Single Perceptron and Batch Perceptron. Tables and Bar Charts for them is given below:

a) All One Initial Weights:

TABLE I
ALL ONE INITIAL WEIGHTS

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

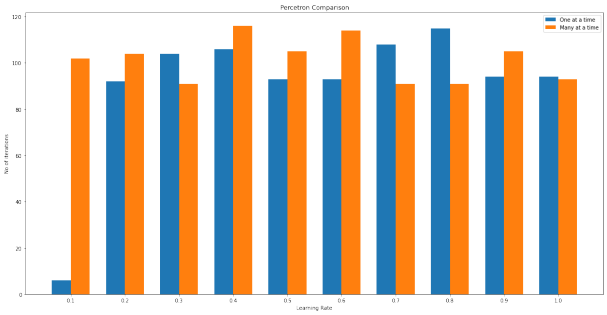

Fig. 3. Bar Chart (All One Initial Weights)

b) All Zero Initial Weights:

TABLE II
ALL ZERO INITIAL WEIGHTS

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	105
0.7	94	105
0.8	94	105
0.9	94	105
1.0	94	92

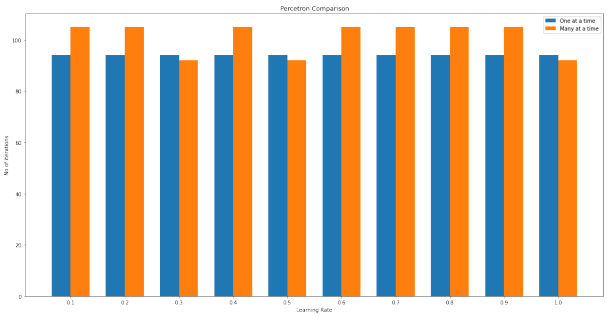

Fig. 4. Bar Chart (All Zero Initial Weights)

c) Randomly Seed Fixed Initial Weights:

TABLE III
RANDOMLY SEED FIXED INITIAL WEIGHTS

Alpha (Learning Rate)	One at a Time	Many at a Time
0.1	97	84
0.2	95	91
0.3	93	117
0.4	101	133
0.5	106	90
0.6	113	105
0.7	94	88
0.8	113	138
0.9	108	138
1.0	101	150

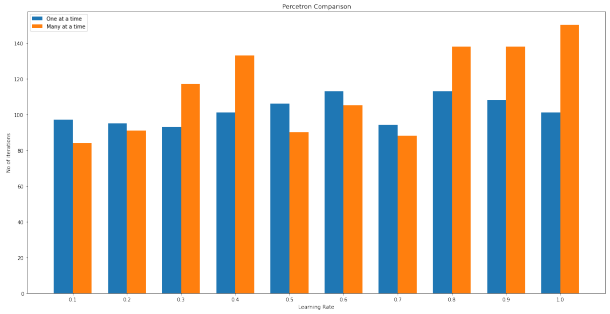

Fig. 5. Bar Chart (Randomly Seed Fixed Initial Weights)

IV. QUESTION ANSWERING

A. We need to take sample points in a higher dimension because when datasets are linear, the decision boundary will work perfectly. If the data is non linear, we will not get the correct hyperplane. Also, they are not linearly separable. For making these points linearly separable we need to take it to a higher dimension.

B. In each of the three initial weight cases for each learning rate the number of updates the algorithm takes before converging is shown in Table I, II and III. Also Figure 3, 4 and 5 shows the visualization of the results.

V. CONCLUSION

In this experiment, we performed both Single and Batch Perceptron.It is clearly seen that, Batch Perceptron takes more time than Single Perceptron. If we used many datasets, there may be possibilities of better results from Batch Perceptron. As Single Perceptron updates itself every time, it is better. Though the solution stops when it gets all datas perfectly classified, the solution space is very large in this case. So, we can say that One at a time approach is better than many at a time.

VI. ALGORITHM IMPLEMENTATION / CODE

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


### Task 1
train_set = []
with open('train-perceptron.txt','r') as file:
    new_reader = csv.reader(file,delimiter=' ')
    for row in new_reader:
        train_set.append(row)

for i in range(len(train_set)):
    for j in range(len(train_set[i])):
        train_set[i][j] = float(train_set[i][j])


a = []
b = []
for train in train_set:
    if train[2]==1:
        a.append([train[0], train[1]])
    elif train[2]==2:
        b.append([train[0], train[1]])
class_a = np.array(a)
class_b = np.array(b)

plt.plot(class_a[:,0:1],class_a[:, 1:], linestyle =
    '', marker='o', color='r', label ="Train Class 1
    ")
plt.plot(class_b[:,0:1],class_b[:, 1:], linestyle =
    '', marker = '*', color = 'k', label ="Train
    Class 2")
plt.legend()
plt.show()


### Task 2
y = np.zeros((6,6))
y = [[class_a[0][0]*class_a[0][0], class_a[0][1]*
    class_a[0][1], class_a[0][0]*class_a[0][1],
    class_a[0][0], class_a[0][1], 1],
    [class_a[1][0]*class_a[1][0], class_a[1][1]*
    class_a[1][1], class_a[1][0]*class_a[1][1],
    class_a[1][0], class_a[1][1], 1],
    [class_a[2][0]*class_a[2][0], class_a[2][1]*
    class_a[2][1], class_a[2][0]*class_a[2][1],
    class_a[2][0], class_a[2][1], 1],

     [-class_b[0][0]*class_b[0][0], -class_b[0][1]*
    class_b[0][1], -class_b[0][0]*class_b[0][1], -
    class_b[0][0], -class_b[0][1], -1],
    [-class_b[1][0]*class_b[1][0], -class_b[1][1]*
    class_b[1][1], -class_b[1][0]*class_b[1][1], -
    class_b[1][0], -class_b[1][1], -1],
    [-class_b[2][0]*class_b[2][0], -class_b[2][1]*
    class_b[2][1], -class_b[2][0]*class_b[2][1], -
    class_b[2][0], -class_b[2][1], -1]
    ]
y = np.array(y)
y


### Task 3

## One at A Time
w = np.ones((1,6))
alpha = 0.1
flag = 0
cnt = 0
g = 0
```

```
while cnt<200:
    cnt = cnt+1
    flag = 0
    for i in range(6):
        g = np.matmul(y[i, :],w.transpose())
        if g<=0:
            w = w+alpha*y[i, :]
        else:
            flag = flag+1
    if flag==6:
        break
print(cnt)


## Many at A Time
w2 = np.ones((1, 6))
alpha2 = 0.1
flag2 = 0
cnt2 = 0
temp = 0
g2 = 0
while cnt2<200:
    cnt2 = cnt2+1
    flag2 = 0
    temp = 0
    for i in range(6):
        g2 = np.matmul(y[i, :], w2.transpose())
        if g2<=0:
            temp = temp + y[i, :]
        else:
            flag2 = flag2+1
    if flag2==6:
        break
    w2 = w2+alpha2*temp
print(cnt2)


### Task 4

## All One
# One at A Time
one_at_ones = []
one_at_alpha = []
for alpha in np.arange(0.1, 1.1, 0.1):
    w = np.ones((1,6))
    cnt = 0
    flag = 0
    g = 0
    while cnt<200:
        cnt = cnt+1
        flag = 0
        for i in range(6):
            g = np.matmul(y[i, :],w.transpose())
            if g<=0:
                w = w+alpha*y[i, :]
            else:
                flag = flag+1
        if flag==6:
            break
    one_at_alpha.append(alpha)
    one_at_ones.append(cnt)

# Many at A Time
many_at_ones = []
for alpha in np.arange(0.1, 1.1, 0.1):
    w2 = np.ones((1, 6))
    alpha2 = 0.1
    flag2 = 0
    cnt2 = 0
    temp = 0
    g2 = 0
    while cnt2<200:
        cnt2 = cnt2+1
        flag2 = 0
        temp = 0
        for i in range(6):
            g2 = np.matmul(y[i, :], w2.transpose())
            if g2<=0:
                temp = temp + y[i, :]
            else:
                flag2 = flag2+1
        if flag2==6:
            break
        w2 = w2+alpha*temp
    many_at_ones.append(cnt2)
df = pd.DataFrame({"Alpha (Learning Rate)":
    one_at_alpha, "One at a Time":one_at_ones, "Many
     at a Time":many_at_ones})
df

labels = ['0.1', '0.2', '0.3', '0.4', '0.5', '0.6',
    '0.7', '0.8', '0.9', '1.0']
one_at_time_ones = one_at_ones
many_at_time_ones = many_at_ones

x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, one_at_time_ones, width
    , label='One at a time')
rects2 = ax.bar(x + width/2, many_at_time_ones,
    width, label='Many at a time')

# Add some text for labels, title and custom x-axis
    tick labels, etc.
ax.set_ylabel('No of iterations')
ax.set_xlabel('Learning Rate')
ax.set_title('Percetron Comparison')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()


def autolabel(rects):
    """Attach a text label above each bar in *rects
    *, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.
    get_width() / 2, height),
                    xytext=(0, 3),  # 3 points
    vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


plt.rcParams["figure.figsize"] = (20,10)

plt.show()


## All Zeros
# One at A Time
one_at_zeros = []
one_at_alpha = []
for alpha in np.arange(0.1, 1.1, 0.1):
    w = np.zeros((1,6))
    cnt = 0
    flag = 0
    g = 0
    while cnt<200:
        cnt = cnt+1
        flag = 0
        for i in range(6):
            g = np.matmul(y[i, :],w.transpose())
            if g<=0:
```

```
                w = w+alpha*y[i, :]
            else:
                flag = flag+1
        if flag==6:
            break
    one_at_alpha.append(alpha)
    one_at_zeros.append(cnt)

# Many at A Time
many_at_zeros = []
for alpha in np.arange(0.1, 1.1, 0.1):
    w2 = np.zeros((1, 6))
    alpha2 = 0.1
    flag2 = 0
    cnt2 = 0
    temp = 0
    g2 = 0
    while cnt2<200:
        cnt2 = cnt2+1
        flag2 = 0
        temp = 0
        for i in range(6):
            g2 = np.matmul(y[i, :], w2.transpose())
            if g2<=0:
                temp = temp + y[i, :]
            else:
                flag2 = flag2+1
        if flag2==6:
            break
        w2 = w2+alpha*temp
    many_at_zeros.append(cnt2)
df = pd.DataFrame({"Alpha (Learning Rate)":
    one_at_alpha, "One at a Time":one_at_zeros, "
    Many at a Time":many_at_zeros})
df

labels = ['0.1', '0.2', '0.3', '0.4', '0.5', '0.6',
    '0.7', '0.8', '0.9', '1.0']
one_at_time_zeros = one_at_zeros
many_at_time_zeros = many_at_zeros

x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, one_at_time_zeros,
    width, label='One at a time')
rects2 = ax.bar(x + width/2, many_at_time_zeros,
    width, label='Many at a time')

# Add some text for labels, title and custom x-axis
    tick labels, etc.
ax.set_ylabel('No of iterations')
ax.set_xlabel('Learning Rate')
ax.set_title('Percetron Comparison')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()


def autolabel(rects):
    """Attach a text label above each bar in *rects
    *, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.
    get_width() / 2, height),
                    xytext=(0, 3),  # 3 points
    vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


plt.rcParams["figure.figsize"] = (20,10)

plt.show()


## Randomly Initialized with Seed Fixed
# One at A Time
one_at_random = []
one_at_alpha = []
for alpha in np.arange(0.1, 1.1, 0.1):
    np.random.seed(10)
    w = np.random.random((1,6))
    cnt = 0
    flag = 0
    g = 0
    while cnt<200:
        cnt = cnt+1
        flag = 0
        for i in range(6):
            g = np.matmul(y[i, :],w.transpose())
            if g<=0:
                w = w+alpha*y[i, :]
            else:
                flag = flag+1
        if flag==6:
            break
    one_at_alpha.append(alpha)
    one_at_random.append(cnt)

# Many at A Time
many_at_random = []
for alpha in np.arange(0.1, 1.1, 0.1):
    np.random.seed(10)
    w2 = np.random.random((1, 6))
    alpha2 = 0.1
    flag2 = 0
    cnt2 = 0
    temp = 0
    g2 = 0
    while cnt2<200:
        cnt2 = cnt2+1
        flag2 = 0
        temp = 0
        for i in range(6):
            g2 = np.matmul(y[i, :], w2.transpose())
            if g2<=0:
                temp = temp + y[i, :]
            else:
                flag2 = flag2+1
        if flag2==6:
            break
        w2 = w2+alpha*temp
    many_at_random.append(cnt2)
df = pd.DataFrame({"Alpha (Learning Rate)":
    one_at_alpha, "One at a Time":one_at_random, "
    Many at a Time":many_at_random})
df

labels = ['0.1', '0.2', '0.3', '0.4', '0.5', '0.6',
    '0.7', '0.8', '0.9', '1.0']
one_at_time_random = one_at_random
many_at_time_random = many_at_random

x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, one_at_time_random,
    width, label='One at a time')
rects2 = ax.bar(x + width/2, many_at_time_random,
    width, label='Many at a time')

# Add some text for labels, title and custom x-axis
    tick labels, etc.
```

```
ax.set_ylabel('No of iterations')
ax.set_xlabel('Learning Rate')
ax.set_title('Percetron Comparison')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()


def autolabel(rects):
    """Attach a text label above each bar in *rects
    *, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.
    get_width() / 2, height),
                    xytext=(0, 3),  # 3 points
    vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


plt.rcParams["figure.figsize"] = (20,10)

plt.show()
```