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

Tahira Salwa Rabbi Nishat
Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204070@aust.edu

***Abstract*—The data is first taken to a higher dimension using the ϕ function.Then is it Normalized. The initial values for the model weights learning rate are set. Then the data is iterated over to get the dot products. if the dot product comes out greater than 0, then data is classified, otherwise weight is updated one by one or after a batch iteration to finally get the data classified.**

***Index Terms*—Perceptron algorithm, batch update, many at a time, one at a time, single update, perceptron normalization, perceptron weight update.**

I. Introduction

The Perceptron Classifier is a linear algorithm that can be applied to binary classification tasks. if some linear data look non linear in a one-dimensional plane, but in a higher dimensional plane, they will look linear and can be classified. So, a ϕ function is introduced. The data are multiplied with the ϕ function. Then the other class is normalized. the data is multiplied with the learning rate then again multiplied with the weight vector, if result is negative, weight is updated and the same thing is done until a positive result is found, if the result is positive, then the data is correctly classified.
For Single Update or One at a time the formula we used:

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

For Batch Update or Many at a time the formula we used:

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

here, η is the learning rate, w is the weight.

II. Experimental Design / Methodology

We were given a data set. The feature was a 6X2 matrix and their class were given. The steps I have followed for this experience are:

1) I have loaded the train file and test file using loadtxt() method from numpy library.
2) I distributed them into two classes according to the value of the given classes.
3) Then I plotted the data using scatter from matplotlib.pyplot in python.
4) Then I took the ϕ function and dot multiplied with the given data of each class.
   $\phi = \begin{bmatrix} x_1^2 & x_2^2 & x_1^2 * x_2^2 & x_1 & x_2 & 1 \end{bmatrix}$

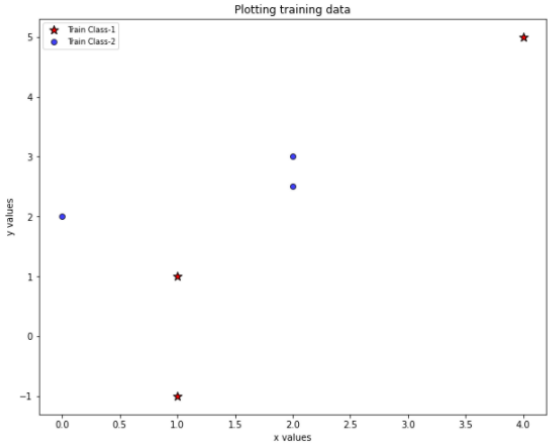

Fig. 1. Plotting train data

5) Then I normalized the data of class2 by multiplying them with -1.
6) Then I initialized three weight vectors, one with all zero values, one with all one values and one with random values.
7) then I initialized learning rate alpha from 0.1 to 1 with a step size of 0.1.
8) After that, I did the batch update for Peceptron algorithm and Single update for Perceptron algorithm for all initial weight values one by one.
9) I took the result from each cases of all initial weight and made a table of the iteration number for batch update and single update.
10) finally plotted the result into a bar diagram for further analysis,

III. Result Analysis

We got the result for the three weight vector. To compare single update result and batch update result, we have plotted them into a bar diagram.

- When all the initial weight is zero the result is, in the single update, all data were converged by the 94^{th} iteration and in batch update, it took highest 105 iterations to get converged.

TABLE I
NUMBER OF ITERATIONS INITIAL WEIGHT VECTOR ALL ZEROS

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

- When all the initial weight is 1 the result is, the single update took highest 115 iterations and batch update took highest 116 iterations to get converged.

TABLE II
NUMBER OF ITERATIONS INITIAL WEIGHT VECTOR ALL ONES

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

- Here we took the initial weight with random values here, the single update took highest 101 iterations and batch update took highest 139 iterations to get converged.

TABLE III
NUMBER OF ITERATIONS INITIAL WEIGHT VECTOR ALL RANDOM VALUES

Alpha (Learning Rate)	One at a Time	Many at a Time
0.1	9	13
0.2	98	108
0.3	91	108
0.4	94	116
0.5	100	89
0.6	87	113
0.7	100	94
0.8	101	139
0.9	99	114
1.0	100	111

Here, the single update took highest 101 iterations and batch update took highest 139 iterations to get converged.

- The bar diagram when all initial weights were zero:

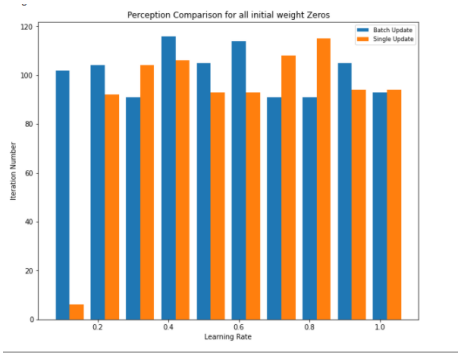

Fig. 2. Table when initial weight was all zeros

- The bar diagram when all initial weights were one:

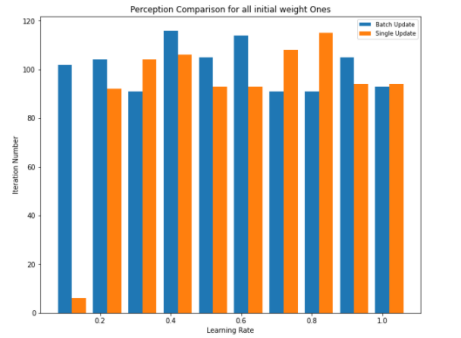

Fig. 3. Table when initial weight was all zeros

- The bar diagram when all initial weights were random values:

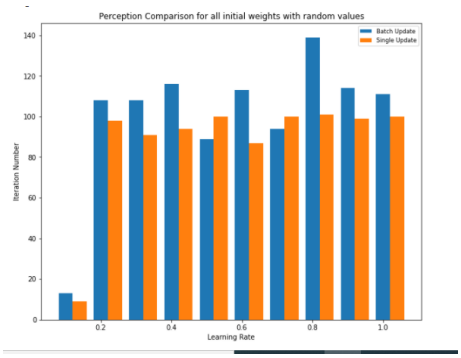

Fig. 4. Table when initial weight was all zeros

IV. QUESTION ANSWER

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

A — if some linear data look non-linear in a one-dimensional plane, in a higher dimensional plane,

they will look linear and can be classified. That's why we need to take the sample points to a higher dimension.

2. Q — In each of the three initial weight cases and for each learning rate, how many updates does the algorithm take before converging?

A — 1. When initial weight was 0, the highest iteration before converging for single update = 94, for batch update = 105
2. When initial weight was 1, the highest iteration before converging for single update = 115, for batch update = 116
3. When initial weight was random values, the highest iteration before converging for single update = 101, for batch update = 139

Ultimately by 150 iteration, all the data were converged in all three cases(Figures are attached in the previous section).

V. Conclusion

In this experiment, we used the Perceptron algorithm both single and batch updates, and both gave nearly identical result. In all cases, for the given data, the algorithm took under 150 iterations before converging.

VI. Algorithm Implementation / Code

```
# -*- coding: utf-8 -*-
"""160204070_Assignment2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
    bb8cngReY4PgKEGxencJNawfnMJzlwkJ
"""

from google.colab import files
uploaded = files.upload()

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

#Reading Train and Test data files
import numpy as np
df_train = np.loadtxt("train-perceptron.txt",dtype=
    float, delimiter = " ")

#Distributing train data into trainClass1 and
    trainClass2 based on the data of the last column

df1 = df_train[np.where(df_train[:,-1]==1)]
w1 =  df1[:, [0,1]]

df2 = df_train[np.where(df_train[:,-1]==2)]

w2 =  df2[:, [0,1]]

df = np.concatenate((w1, w2), axis=0)

print(' lass1 =  \n',w1)
print(' lass2 =  \n',w2)
print('data=  \n',df)

#Plotting train data
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, figsize = (10, 8))


plt.scatter(w1[:,0], w1[:,1], c = 'red', marker='*',
     s= 100, label='Train Class-1', edgecolors='
    black')
plt.scatter(w2[:,0], w2[:,1], c = 'blue', label='
    Train Class-2', edgecolors='black', alpha=0.75)

plt.title("Plotting training data")
ax.legend(frameon=True, shadow=True, borderpad=1)
plt.xlabel("x values")
plt.ylabel("y values")
plt.legend(loc="best",fontsize="small")

plt.show()

y= np.empty(shape=[0, 6])

#multyplyting trainclass1 with given formula to
    acheive higher dimention
for i in range(0,df_train.shape[0]):
    m = np.array([[df[i, 0] * df[i, 0], df[i, 1] *
    df[i, 1], df[i, 0] * df[i, 1], df[i, 0], df[i,
    1], 1]]);
    y = np.append(y, m, axis=0)

print(y)

#Normalization of the second class
y=np.concatenate((y[0:np.shape(w1)[0],:],-1*y[np.
    shape(w1)[0]:,:]),axis=0)
print(y)

weight_0 = np.array([[0,0,0,0,0,0]])

weight_1 = np.array([[1,1,1,1,1,1]])
weight_rand = np.round([10*np.array(np.random.rand
    (6))])
print(weight_0, weight_1, weight_rand)
np.random.seed(0) ;
# weight3 = np.random.rand(6)
# print(weight3)
alpha=np.array
    ([0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0])

#Batch update with initial weight 0
c = 0
batch_0_iter = []
for i in alpha:
  a = np.array(weight_0)
  c = 0
  while True:
    weight_update = []
    tmp = []
    b = 0
    for j in y:
        s = np.sum(j * a,axis = 1)
        if s[0]<= 0:
          tmp.append(j)
          b = 1
    c = c + 1
    if b == 0:
      break
    e = np.array(tmp)
    weight_update.append(np.sum(e, axis = 0))
    weight_update = np.array(weight_update)
    weight_update = a + (weight_update * i)
    a = weight_update
  batch_0_iter.append(c)
```

```
#Single update with initial weight 0
c = 0
single_0_iter = []
for i in alpha:
  a = np.array(weight_0)
  c = 0
  while True:
    b = 0
    for j in y:
        s = np.sum(j * a,axis = 1)
        if s[0]<= 0:
          a = a + (j * i)
          b = 1
    c = c + 1
    if b == 0:
      break

  single_0_iter.append(c)

print("Case 1:Number of iteratons Initial Weight
    Vector All Zeros")
dataset1 = pd.DataFrame({'Alpha': alpha, 'Single
    Update': single_0_iter,'Batch Update':
    batch_0_iter})
print(dataset1.to_string(index=False))

plt.figure(1)
fig, ax = plt.subplots(1, figsize = (10, 8))

plt.bar(alpha, batch_0_iter, 0.04, label='Batch
    Update')
plt.bar(alpha+0.04, single_0_iter, 0.04, label='
    Single Update')

plt.title('Perception Comparison for all initial
    weight Zeros')
plt.xlabel("Learning Rate")
plt.ylabel("Iteration Number")
plt.legend(loc="best",fontsize="small")

#Batch update with initial weight 1
c = 0
batch_1_iter = []
for i in alpha:
  a = np.array(weight_1)
  c = 0
  while True:
    weight_update = []
    tmp = []
    b = 0
    for j in y:
        s = np.sum(j * a,axis = 1)
        if s[0]<= 0:
          tmp.append(j)
          b = 1
    c = c + 1
    if b == 0:
      break
    e = np.array(tmp)
    weight_update.append(np.sum(e, axis = 0))
    weight_update = np.array(weight_update)
    weight_update = a + (weight_update * i)
    a = weight_update
  batch_1_iter.append(c)

#Single update with initial weight 1
c = 0
single_1_iter = []
for i in alpha:
  a = np.array(weight_1)
  c = 0
  while True:
    b = 0
    for j in y:
        s = np.sum(j * a,axis = 1)
        if s[0]<= 0:
          a = a + (j * i)
          b = 1
    c = c + 1
    if b == 0:
      break

  single_1_iter.append(c)

print("Case 2: Number of iteratons Initial Weight
    Vector All Ones")
dataset1 = pd.DataFrame({'Alpha': alpha, 'Single
    Update': single_1_iter,'Batch Update':
    batch_1_iter})
print(dataset1.to_string(index=False))

plt.figure(2)
fig, ax = plt.subplots(1, figsize = (10, 8))

plt.bar(alpha, batch_1_iter, 0.04, label='Batch
    Update')
plt.bar(alpha+0.04, single_1_iter, 0.04, label='
    Single Update')

plt.title('Perception Comparison for all initial
    weight Ones')
plt.xlabel("Learning Rate")
plt.ylabel("Iteration Number")
plt.legend(loc="best",fontsize="small")

#Batch update with initial weight all random values
c = 0
batch_rand_iter = []
for i in alpha:
  a = np.array(weight_rand)
  c = 0
  while True:
    weight_update = []
    tmp = []
    b = 0
    for j in y:
        s = np.sum(j * a,axis = 1)
        if s[0]<= 0:
          tmp.append(j)
          b = 1
    c = c + 1
    if b == 0:
      break
    e = np.array(tmp)
    weight_update.append(np.sum(e, axis = 0))
    weight_update = np.array(weight_update)
    weight_update = a + (weight_update * i)
    a = weight_update
  batch_rand_iter.append(c)

#Single update with initial random values
c = 0
single_rand_iter = []
for i in alpha:
  a = np.array(weight_rand)
  c = 0
  while True:
    b = 0
    for j in y:
        s = np.sum(j * a,axis = 1)
        if s[0]<= 0:
          a = a + (j * i)
          b = 1
    c = c + 1
    if b == 0:
      break
```

```
  single_rand_iter.append(c)

print("Case 3:Number of iteratons  Initial Weight
    Vector All Random values")
dataset1 = pd.DataFrame({'Alpha': alpha, 'Single
    Update': single_rand_iter,'Batch Update':
    batch_rand_iter})
print(dataset1.to_string(index=False))

plt.figure(2)
fig, ax = plt.subplots(1, figsize = (10, 8))

plt.bar(alpha, batch_rand_iter, 0.04, label='Batch
    Update')
plt.bar(alpha+0.04, single_rand_iter, 0.04, label='
    Single Update')

plt.title('Perception Comparison for all initial
    weights with random values')
plt.xlabel("Learning Rate")
plt.ylabel("Iteration Number")
plt.legend(loc="best",fontsize="small")

fig, axs = plt.subplots(3, sharex=True, sharey=True,
     figsize = (10, 8))
fig.suptitle('Perceptron Comperison for weight all 0
    s, all 1s and all randoms')


axs[0].bar(alpha, batch_0_iter, 0.04, label='Batch
    Update')
axs[0].bar(alpha+0.04, single_0_iter, 0.04, label='
    Single Update')


axs[1].bar(alpha, batch_1_iter, 0.04, label='Batch
    Update')
axs[1].bar(alpha+0.04, single_1_iter, 0.04, label='
    Single Update')

axs[2].bar(alpha, batch_rand_iter, 0.04, label='
    Batch Update')
axs[2].bar(alpha+0.04, single_rand_iter, 0.04, label
    ='Single Update')


plt.xlabel("Learning Rate")
plt.ylabel("Iteration Number")
plt.legend(loc="best",fontsize="small")
```