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

Rakib Hossain Rifat
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204099@aust.edu

***Abstract*—Main objective of this experiment is finding optimal weight using perception algorithm of Linear Discriminant function.**

***Index Terms*—Perception, Update Rule,Linear Discriminant function .**

I. Introduction

Perception algorithm is used to find the optimal weights for a specific learning rate so that all the data points can classify correctly.

Using update rule optimal weights are calculated.

II. Methodology

In this experiment I need to perform several tasks.

A. Plotting Training Data of Both Classes

for this I need to plot training data of both class from "train.txt", and samples belongs to same class should have same marker and color and observe if both classes can be separated with a liner boundary.

B. Generate High Dimensional Data Points

for this I need to use the phi function given bellow.

$$Y = [X_1^2 X_2^2 X_1 * X_2 X_1 X_2 1]$$

Perception Algorithm is created for linear data , so we need to convert data points in higher order to convert non linear data to linear data . This is why i need to take the sample points to a high dimension.

C. Updating weight using update rule

for doing this task i have to check if weights are enough to classify data points correctly otherwise update them using update rule.

$$\underline{w}(i+1) = \underline{w}(i) + \alpha * \sum y$$

if

$$\underline{w}^T(i) * y < 0$$

[Batch Update]

$$\underline{w}(i+1) = \underline{w}(i) + \alpha * y$$

[Single Update]

$$\underline{w}(i+1) = \underline{w}(i)$$

if

$$\underline{w}^T(i) * y > 0$$

D. Calculating Weights

for this I need to calculate weights that can classify all data points correctly with initial weight all zero , all one , random weight with fixed seed .

when all zero, initial weights are [0.0,0.0,0.0,0.0,0.0,0.0] when all one, initial weights are [1.0,1.0,1.0,1.0,1.0,1.0] and with fixed seed 6.

this task will be done using both many at a time and one at a time approach. And result will be shown in table and bar chart. In each of the three initial weight cases and for each learning rate, how many updates the algorithm take before converging is given in table and bar chart in result analysis.

III. Result Analysis

A. Plotting Training Data of Both Classes

Here i have plotted data of both class with different marker and color.
Class 1= [(1, 1.0), (1, -1.0), (4, 5.0)]
Class 2= [(2, 2.5), (0, 2.0), (2, 3.0)]

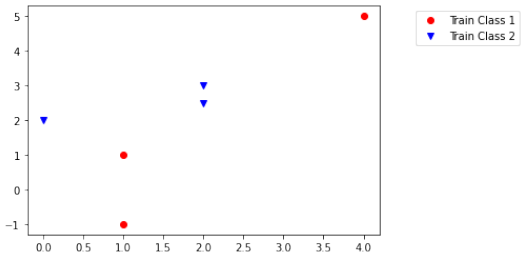

Fig. 1. Training Data of Class One and Class Two

now, if i try to separate both class using linear decision boundary then it shows that that is not possible with any line. for drawing line,

$$Y = mx + c$$

formula is used here

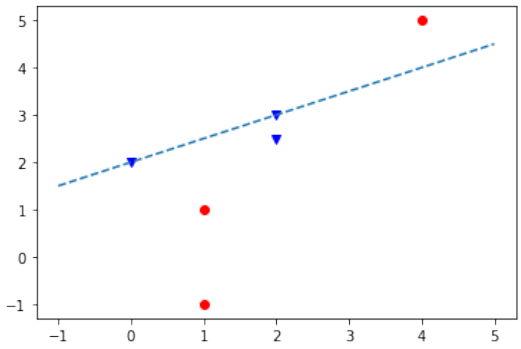

Fig. 2. Training Data of Class One and Class Two with Linear Decision Boundary

B. Generating the high dimensional sample points and Normalize

Using phi function high dimensional data points are generated and omega two is normalized.

C. Perception Comparison when initial weights are zero

When initial weights are zero we have compared iteration needed in both methods and with different learning rate.

TABLE I
PERCEPTION COMPARISON WHEN INITIAL WEIGHTS ARE ZERO

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
0.4	94	92
0.6	94	105
0.7	94	105
0.8	94	105
0.9	94	105
1.0	94	92

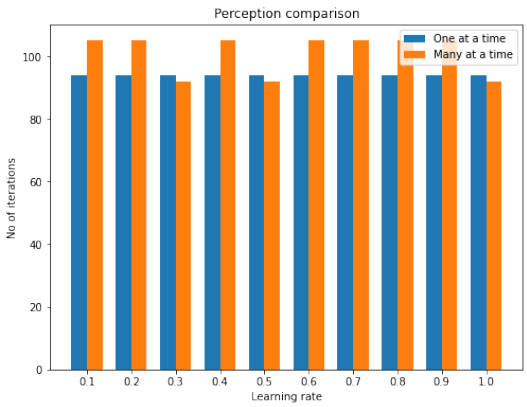

Fig. 3. Perception Comparison when initial weights are zero

D. Perception Comparison when initial weights are one

When initial weights are one we have compared iteration needed in both methods and with different learning rate.

TABLE II
PERCEPTION COMPARISON WHEN INITIAL WEIGHTS ARE ONE

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
0.4	93	105
0.6	93	114
0.7	108	91
0.8	115	91
0.9	94	105
1.0	94	93

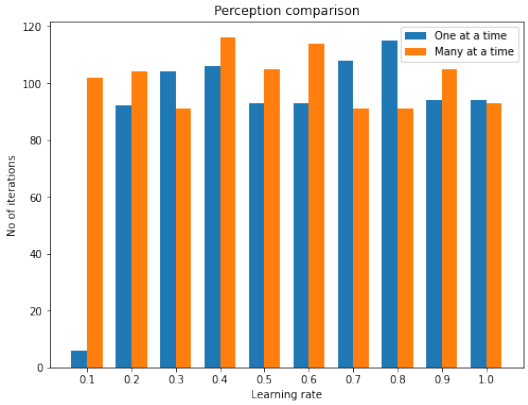

Fig. 4. Perception Comparison when initial weights are one

E. Perception Comparison when initial weights are random

When initial weights are random we have compared iteration needed in both methods and with different learning rate.

TABLE III
PERCEPTION COMPARISON WHEN INITIAL WEIGHTS ARE RANDOM

Learning rate	*One at a time*	*Many at a time*
0.1	31	64
0.2	15	13
0.3	89	109
0.4	90	114
0.4	6	116
0.6	87	110
0.7	88	104
0.8	102	88
0.9	94	4
1.0	107	104

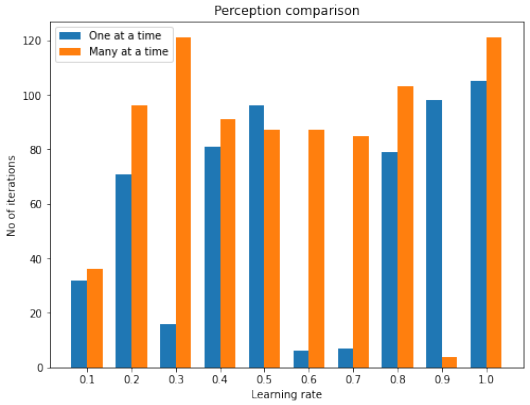

Fig. 5. Perception Comparison when initial weights are random

F. Result Comparison

From the bar chart and table we can see that for different learning rate algorithm taking different times of iteration . Even using same learning rate but different approach is taking different times of iteration. when we are taking many at a time update random weight with learning rate 0.9 is taking minimum number of iteration . And for one at a time all weight one and learning rate 0.1 and random weight with learning rate 0.5 is taking minimum number of iteration to classify all data points correctly .

IV. Conclusion

From the experiment it can be state that learning rate and approach plays an vital role in classifying data points. So tuning hyper parameters correctly can boost up the speed of the algorithm.

V. Code

```
# -*- coding: utf-8 -*-
"""160204099_B2_02.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
    NEAGkCgul82_bQVj3Au-T06mefUR2XwU
"""

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

"""Variables"""

omega_one=[]
omega_two=[]
omega_one_phi=[]
omega_two_phi=[]
lr = np.arange(0.1,1.1,0.1)

"""Upload And path setup"""

train_df=pd.read_csv("/content/train-perceptron.txt"
    ,sep=" " ,names=["x1","x2","Y"])

"""Traning Data Separation"""

def class_separation(df):
  for i in range(0,len(df)):
    li=[]
    if df['Y'][i]==1:
      li=df['x1'][i],df['x2'][i]
      omega_one.append(li)
    elif df['Y'][i]==2:
      li=df['x1'][i],df['x2'][i]
      omega_two.append(li)

"""Many at a Time"""

def batch_update(learning_rate,weights):
    correctly_classified=False
    miss_classified=0
    iteration=0
    while correctly_classified==False and iteration
    <=200:
      miss_classified = 0
      sum_weights=np.zeros(6)
      for i in range(len(phi)):
        if (np.dot(weights,phi[i])) <= 0:
              sum_weights = np.add(sum_weights , phi
    [i])
              miss_classified +=1
      if miss_classified == 0:
        correctly_classified = True
      else:
        correctly_classified=False
        weights = weights + np.dot(learning_rate ,
    sum_weights)
      iteration += 1

    return iteration

"""One at a Time"""

def single_update(learning_rate,weights):
    correctly_classified=False
    iteration=0
    while correctly_classified==False and iteration
    <=200:
      miss_classified=0
      for i in range(len(phi)):
        if (np.dot(weights,phi[i])) <= 0:
              weights = weights + np.dot(
    learning_rate,phi[i])
              miss_classified +=1
      if miss_classified == 0:
        correctly_classified = True
      iteration += 1

    return iteration

"""Phi Function Claculation"""

def phi(omega_value):
  return [pow(omega_value[0],2),pow(omega_value
    [1],2),omega_value[0]*omega_value[1],omega_value
    [0],omega_value[1],1]

"""Table Show"""

def table_show(learning_rate,many_at_a_time,
    one_at_a_time):

  dict = {'Learning rate': learning_rate, 'Many at a
     time': many_at_a_time, 'One at a time':
    one_at_a_time}
  dataframe = pd.DataFrame(dict)

  print(dataframe.to_string(index=False))
```

```
"""Perception comparison"""

def perception_comparison(learning_rate,
    many_at_a_time,one_at_a_time):
  labels=[]
  for i in learning_rate:
    labels.append(str(round(i,2)))
  x = np.arange(len(lr))
  width = 0.35
  fig, ax = plt.subplots()

  fig.set_size_inches(8,6)
  ax.bar(x - width/2, one_at_a_time, width, label='
    One at a time')
  ax.bar(x + width/2, many_at_a_time, width, label='
    Many at a time')

  ax.set_xlabel('Learning rate')
  ax.set_ylabel('No of iterations')
  ax.set_title('Perception comparison')
  ax.set_xticks(x)
  ax.set_xticklabels(labels)
  ax.legend()

  plt.show()

"""Separating Class"""

class_separation(train_df)

"""Plotting Classes without Decision Boundary"""

plt.plot(*zip(*omega_one),'ro',label="Train Class 1"
    )
plt.plot(*zip(*omega_two),"bv",label="Train Class 2"
    )
plt.legend(loc="upper center",bbox_to_anchor=(1.25,
    1))
plt.show()

"""Plotting Classes with Decision Boundary"""

minimum=min(min(train_df["x1"]),min(train_df["x2"]))
maximum=max(max(train_df["x1"]),max(train_df["x2"]))
x=[]
y=[]
m=0.5
c=2
for i in range(int(minimum),int(maximum+1)):
  x.append(i)
  y.append(m*i+c)

plt.plot(*zip(*omega_one),'ro',label="Train Class 1"
    )
plt.plot(*zip(*omega_two),"bv",label="Train Class 2"
    )
plt.plot(x,y,"--",label="Decision Boundary")

plt.show()

"""Generate the high dimensional sample points"""

for omega in omega_one:
  omega_one_phi.append(phi(omega))
for omega in omega_two:
  omega_two_phi.append(phi(omega))
normalize_omega_two=np.array(omega_two_phi).dot(-1)
phi = np.concatenate((np.array(omega_one_phi),
    normalize_omega_two))

"""All Weights Zero"""

single_iter=[]
weights=np.zeros(6)
for i in lr:
  iter= single_update(i,weights)
  single_iter.append(iter)
batch_iter=[]
for i in lr:
  iter= batch_update(i,weights)
  batch_iter.append(iter)
table_show(lr,batch_iter,single_iter)

perception_comparison(lr,batch_iter,single_iter)

"""All Weights One"""

single_iter=[]
weights=np.ones(6)
for i in lr:
  iter= single_update(i,weights)
  single_iter.append(iter)
batch_iter=[]
for i in lr:
  iter= batch_update(i,weights)
  batch_iter.append(iter)
table_show(lr,batch_iter,single_iter)

perception_comparison(lr,batch_iter,single_iter)

"""Random Weights"""

single_iter=[]
weights=[]
np.random.seed(6)
for i in range(6):
  weights.append(random.randint(0, 6))
for i in lr:
  iter= single_update(i,weights)
  single_iter.append(iter)
batch_iter=[]
for i in lr:
  iter= batch_update(i,weights)
  batch_iter.append(iter)
table_show(lr,batch_iter,single_iter)

perception_comparison(lr,batch_iter,single_iter)
```