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

Md. Yeasir Arafat
Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204093@aust.edu

***Abstract*—In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function that can decide whether or not an input belongs to some specific class.**

***Index Terms*—Keep changing the orientation of the hyperplane until all training samples are on its positive side.**

I. Introduction

A perceptron, a neuron's computational prototype, is categorized as the simplest form of a neural network. Frank Rosenblatt invented the perceptron at the Cornell Aeronautical Laboratory in 1957. A perceptron has one or more than one inputs, a process, and only one output.

A linear classifier that the perceptron is categorized as is a classification algorithm, which relies on a linear predictor function to make predictions. Its predictions are based on a combination that includes weights and feature vectors. The linear classifier suggests two categories for the classification of training data.

II. Experimental Design / Methodology

1. **Plotting all sample points:** Two sample classes are given to 'train.txt' file. We need to plot all the sample points of both classes using different markers. First I read the file from google drive then extract the data for plotting. Finally I plotted the points using matplotlib package. And I observed that these two classes is not linearly separable.

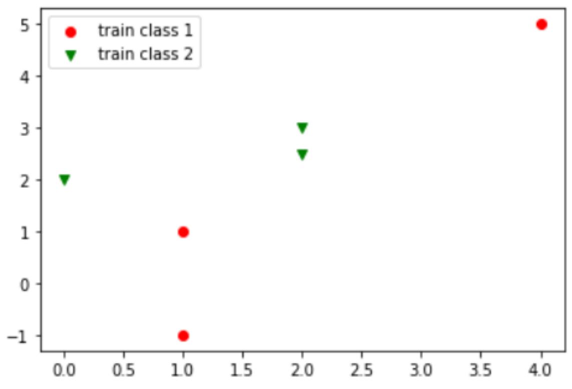

Fig 1: Sample points of Train Data

2. **Generating high dimension sample points:** As given sample points are not linearly separable, we use given phi function to convert the 2D data to 6D data and then it will be linearly separable. Then we normalize class 2, simply multiplying high dimensional class 2 points with -1.

3. **Finding weights using Perceptron algorithm:** Now we need to find a specific weights which could correctly classify all the data points. For this we need to update weights in each iteration using perceptron algorithm until all the sample points are classified with that weight.

$$a(k+1) = a(k) + \eta(\sum_{\forall y \; missclassified} y) \quad (1)$$
$$if\; W^T y <= 0 \; then\; missclassified$$

We will update weights with this perceptron equation for batch update and single update.

4. **Testing:** I used three different types of initial weights(all one, all zero, randomly initialized with seed fixed). For all of these three cases I varied the learning rate between 0.1 to 1 with step size 0.1.

(a) As given sample points are not linearly separable, we need to use a phi function to convert the low dimension data to higher dimension data as it would be linearly separable.

(b) No of updates does the algorithm take before converging is shown in the below tables:

No of updates for Initial Weight Vector All One:

Learning Rate	One at a time	Many at a time
0.1	6	102
0.2	147	104
0.3	149	191
0.4	149	116
0.5	141	105
0.6	157	114
0.7	136	91
0.8	136	91
0.9	140	105
1.0	141	93

No of updates for Initial Weight Vector All Zero:

Learning Rate	One at a time	Many at a time
0.1	141	105
0.2	141	105
0.3	141	105
0.4	141	105
0.5	141	92
0.6	141	105
0.7	141	92
0.8	141	105
0.9	141	105
1.0	141	92

No of updates for Randomly selected Weights:

Learning Rate	One at a time	Many at a time
0.1	75	89
0.2	34	60
0.3	25	84
0.4	105	19
0.5	20	88
0.6	112	24
0.7	128	18
0.8	144	11
0.9	138	15
1.0	8	104

III. Result Analysis

Plotting Bar chart to analyze the result:

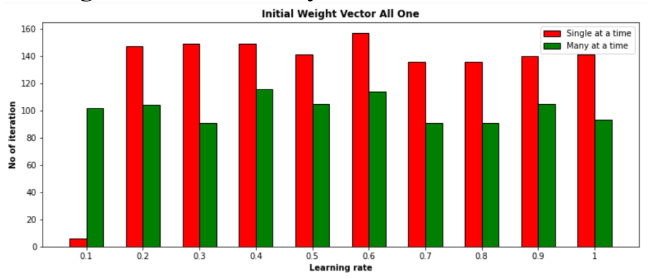

Fig: Bar plot weights all one

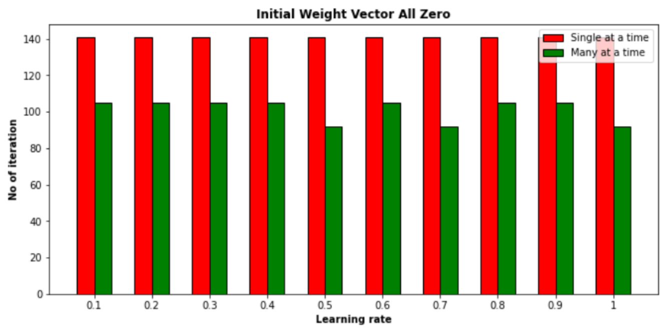

Fig: Bar plot weights all zero

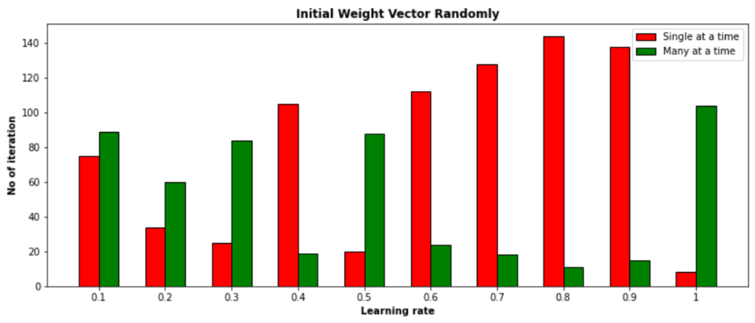

Fig: Bar plot for random weights

From the bar chart we can see, only 6 iteration is needed for this dataset if weights are all one and learning rate is 0.1 and if we use single update. But most of the cases we can see batch update takes less iteration to train the dataset. So we should use batch update if we have enough available memory.

IV. Conclusion

Perceptron today has become an important learning algorithm in the world of artificial intelligence and machine learning. It is considered a reliable and fast solution for the category of problems it has the capabilities of solving. Also, if we develop an understanding of how the perceptron works, we will find the job of understanding more complex networks a lot easier.

V. Algorithm Implementation / Code

```
# File Read from gdrive

from google.colab import drive
drive.mount('/content/gdrive')
# change working directory on the drive
%cd '/content/gdrive/My Drive/Data/'

# read train.txt file line by line
with open('train-perceptron.txt', "r") as train:
    FileasList = train.readlines()

# split the string and store it into another list
    classwise
train = []
train_x1 = []
train_y1 = []
train_x2 = []
train_y2 = []
for i in range(len(FileasList)):
  train.append(FileasList[i].split())
  if(train[i][2] == '1'):
    train_x1.append(float(train[i][0]))
    train_y1.append(float(train[i][1]))
  else:
    train_x2.append(float(train[i][0]))
    train_y2.append(float(train[i][1]))

"""Plotting all sample points """

import matplotlib.pyplot as plt

plt.scatter(train_x1, train_y1, c = 'r', marker = 'o
    ', label = 'train class 1')
plt.scatter(train_x2, train_y2, c = 'g', marker = 'v
    ', label = 'train class 2')
plt.legend(loc = 'best')
```

```
"""Transform the 2D features to 6D using Phi
    Function & normalizing class 2"""

import numpy as np

data = np.empty((0,6), float)
for i in range(len(train)):
  x1 = float(train[i][0])
  x2 = float(train[i][1])
  cls = float(train[i][2])
  if(cls == 1):
    data = np.append(data, [[x1**2, x2**2, x1*x2, x1
    , x2, 1]], axis = 0)
  else:
    data = np.append(data, [[-x1**2, -x2**2, -x1*x2,
     -x1, -x2, -1]], axis = 0)

print(data)

"""Batch Update"""

def batchUpdate(iteration, weights, learningRate):
  w = np.array(weights,dtype = float)
  for i in range(iteration):
    temp = np.full_like(w, 0)
    for features in data:
      g1 =  np.dot(w, features)
      # if missclassified
      if(g1 <= 0):
        temp = temp + features
    # Update weights
    w = w + learningRate*temp

    # all weights are classified break loop
    if(np.count_nonzero(temp) == 0):
      break
  return (w,i+1)

"""Single Update"""

def singleUpdate(iteration, weights, learningRate):
  w = np.array(weights,dtype = float)
  for i in range(iteration):
    count = 0
    for features in data:
      g1 =  np.dot(w, features)
      # if missclassified
      if(g1 <= 0):
        # Update Weights
        count = count + 1
        w = w + learningRate*features
    # all weights are classified break loop
    if(count == 0):
      break
  return (w,i+1)


"""**Initial Weight Vector All One**"""

iterations = 200
learningRate = [.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
weights = [1,1,1,1,1,1]

# batch update weights all one
b_w1_itr = []
b_w1_w = []

for lr in learningRate:
  w,i = batchUpdate(iterations, weights, lr)
  b_w1_itr.append(i)
  b_w1_w.append(w)

# single update weights all one
s_w1_itr = []
s_w1_w = []

for lr in learningRate:
  w,i = singleUpdate(iterations, weights, lr)
  s_w1_itr.append(i)
  s_w1_w.append(w)


# Plotting Data

x = np.arange(len(learningRate))  # the label
    locations
width = 0.3  # the width of the bars

fig, ax = plt.subplots(figsize =(14, 5))
ax.bar(x - width/2, s_w1_itr, width, label='Single
    at a time',color ='r', edgecolor ='black')
ax.bar(x + width/2, b_w1_itr, width, label='Many at
    a time',color ='g', edgecolor ='black')

# Add some text for labels, title and custom x-axis
    tick labels, etc.
ax.set_xlabel('Learning rate', fontweight ='bold')
ax.set_ylabel('No of iteration', fontweight ='bold')
ax.set_title('Initial Weight Vector All One',
    fontweight ='bold')
ax.set_xticks(x)
ax.set_xticklabels(learningRate)
ax.legend()


# Table
import pandas as pd
df = pd.DataFrame(index=learningRate)
df = pd.concat([df, pd.DataFrame(s_w1_itr, index=
    learningRate, columns=['One at a time'])], axis
    =1)
df = pd.concat([df, pd.DataFrame(b_w1_itr, index=
    learningRate, columns=['Many at a time'])], axis
    =1)

def highlight_min(s):
    '''
    highlight the minimum in a Series yellow.
    '''
    is_min = s == s.min()
    return ['background-color: yellow' if v else ''
    for v in is_min]

df.style.apply(highlight_min, subset=['One at a time
    ','Many at a time'])
#print(df.loc[.1]['One at a time'])

"""**Initial Weight Vector All Zero**"""

iterations = 200
learningRate = [.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
weights = [0,0,0,0,0,0]

# batch update weights all one
b_w1_itr = []
b_w1_w = []

for lr in learningRate:
  w,i = batchUpdate(iterations, weights, lr)
  b_w1_itr.append(i)
  b_w1_w.append(w)

# single update weights all one
s_w1_itr = []
s_w1_w = []

for lr in learningRate:
  w,i = singleUpdate(iterations, weights, lr)
  s_w1_itr.append(i)
  s_w1_w.append(w)


# Plotting Data
x = np.arange(len(learningRate))  # the label
    locations
width = 0.3  # the width of the bars

fig, ax = plt.subplots(figsize =(11, 5))
ax.bar(x - width/2, s_w1_itr, width, label='Single
    at a time',color ='r', edgecolor ='black')
ax.bar(x + width/2, b_w1_itr, width, label='Many at
    a time',color ='g', edgecolor ='black')

# Add some text for labels, title and custom x-axis
    tick labels, etc.
ax.set_xlabel('Learning rate', fontweight ='bold')
ax.set_ylabel('No of iteration', fontweight ='bold')
ax.set_title('Initial Weight Vector All Zero',
    fontweight ='bold')
ax.set_xticks(x)
ax.set_xticklabels(learningRate)
ax.legend()

# Table
df = pd.DataFrame(index=learningRate)
df = pd.concat([df, pd.DataFrame(s_w1_itr, index=
    learningRate, columns=['One at a time'])], axis
    =1)
df = pd.concat([df, pd.DataFrame(b_w1_itr, index=
    learningRate, columns=['Many at a time'])], axis
    =1)

df.style.apply(highlight_min, subset=['One at a time
    ','Many at a time'])

"""**Initialized Weight Vector Randomly**"""

import random
iterations = 200
learningRate = [.1,.2,.3,.4,.5,.6,.7,.8,.9,1]

random.seed(1)
weights = [random.randint(1,5),random.randint(6,10),
    random.randint(11,15),random.randint(16,20),
    random.randint(5,15),random.randint(10,20)]

# batch update weights all one
b_w1_itr = []
b_w1_w = []

for lr in learningRate:
  w,i = batchUpdate(iterations, weights, lr)
  b_w1_itr.append(i)
  b_w1_w.append(w)

# single update weights all one
s_w1_itr = []
s_w1_w = []

for lr in learningRate:
  w,i = singleUpdate(iterations, weights, lr)
  s_w1_itr.append(i)
  s_w1_w.append(w)


# Plotting Data
x = np.arange(len(learningRate))  # the label
    locations
width = 0.3  # the width of the bars

fig, ax = plt.subplots(figsize =(13, 5))
ax.bar(x - width/2, s_w1_itr, width, label='Single
    at a time',color ='r', edgecolor ='black')
ax.bar(x + width/2, b_w1_itr, width, label='Many at
    a time',color ='g', edgecolor ='black')

# Add some text for labels, title and custom x-axis
    tick labels, etc.
ax.set_xlabel('Learning rate', fontweight ='bold')
ax.set_ylabel('No of iteration', fontweight ='bold')
ax.set_title('Initial Weight Vector Randomly',
    fontweight ='bold')
ax.set_xticks(x)
ax.set_xticklabels(learningRate)
ax.legend()

# Table
df = pd.DataFrame(index=learningRate)
df = pd.concat([df, pd.DataFrame(s_w1_itr, index=
    learningRate, columns=['One at a time'])], axis
    =1)
df = pd.concat([df, pd.DataFrame(b_w1_itr, index=
    learningRate, columns=['Many at a time'])], axis
    =1)

df.style.apply(highlight_min, subset=['One at a time
    ','Many at a time'])
```