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

Md. Ashraful Haque
*dept. Computer Science and Engineering*
*Ahsanullah University of Science and Technology*
Dhaka, Bangladesh
160204101@aust.edu

***Abstract*—Perceptron is one of the most famous first generation supervised learning algorithm for binary classification. It was later developed into gradient descent with some additional modifications.**

## I. Introduction

The primary objective of the algorithm is to fit the decision boundary to a spot from where almost all the train data are correctly classified. This is done by adjusting the weight vector through trial and error.

The basic components of the algorithm are the input vector, the weight vector, Learning Rate. There are 2 basic ways of executing the algorithm. First one is the Batch technique or Many at a time and the second one is the single technique or one at a time. Different circumstances can give priority of one of the two over other.

## II. Experimental Design / Methodology

As it is an experiment of finding the optimal weights for the discriminant function, only the training part of the data set is needed.

The components that are needed for this experiment are **Weight vector** which is generally initialized with zeros or ones, an **input vector** which is of the same dimension as the weight vector, a **learning rate**. The equation that we will follow is,

**step 1**

Determine whether the data is linearly separable or not. If not then a $fai$ function can be used to take the data into higher dimension to make it linearly separable.

$$y = [x_1^2, x_2^2, x_1 * x_2, x_1, x_2, 1]$$

**step 2**

Normalize any one of the two classes of the data set. This done by simply negating any any the two classes.

lets say the updated input vector is $y$ . Noe the decision rule for identifying the correctly classified entries will be,

if, $w^t y > 0$ then correctly classified
else misclassified

**step 3**

The following equations should be used to find the optimal weight vectors.

**Many at a time,**

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

**One at a time,**

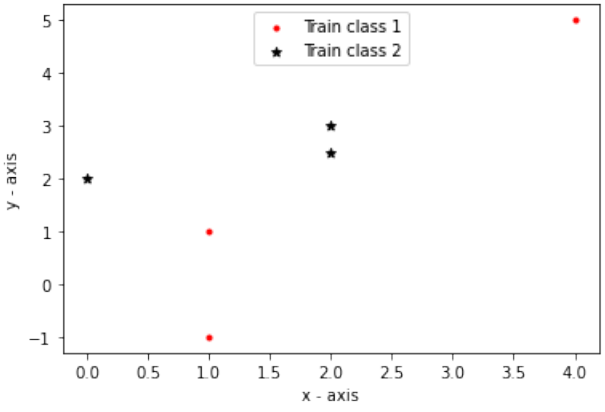
$$w(t+1) = w(t) + \eta(y_{misclassified})$$

here,
$w(t) =$ weight vector at a time $t$
$w(t+1) =$ updated weight vector at a time $t$
$\eta =$ Learning Rate
$y =$ Input Vector
The Algorithm That we will follow is as follows,
Our experiments will be conducted by making some variations in learning rate and initialization of the weight vector.

## III. Result Analysis

**Task 1**

After plotting the training data, it is observed that the data is not linearly separable. Therefore, the data should be transferred to a higher dimension in order to make them linearly separable. **(ans. to question a)**



**Task 2**

The data has been translated using the fai function into 6th dimension making it a linearly separable dataset.

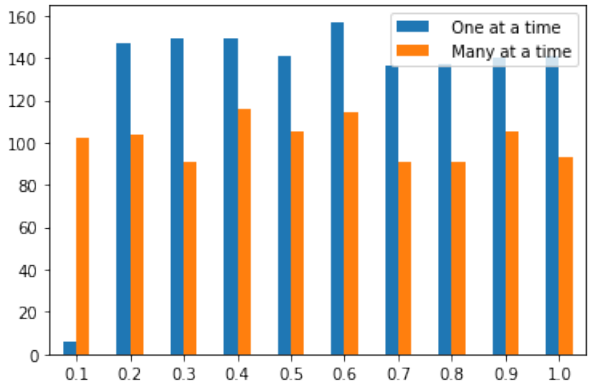
**Task 3**

After using perceptron we found the weight coefficients of the discriminant function. Here

**Weight coefficients has been set to 1 by default**. Number of iterations will go upto 200. (**Setup 1**)

| Learning Rate | One at a time | Many at a time |
|---|---|---|
| .1 | 6 | 102 |
| .2 | 147 | 104 |
| .3 | 149 | 91 |
| .4 | 149 | 116 |
| .5 | 141 | 105 |
| .6 | 157 | 114 |
| .7 | 136 | 91 |
| .8 | 137 | 91 |
| .9 | 140 | 105 |
| 1.0 | 141 | 93 |

**Bar chart:**



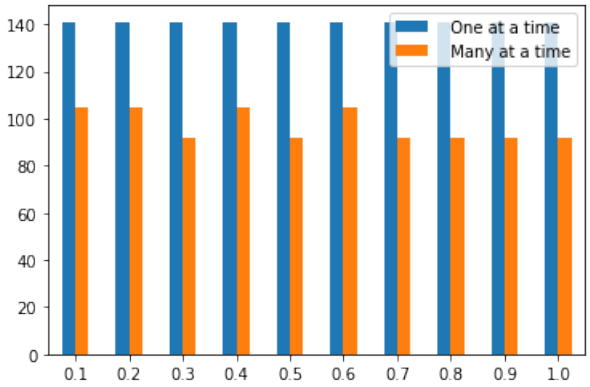

**Task 4**

The experiment has been extended with the following variations.

**Weight coefficients has been set to 0 by default**. Number of iterations will go up to 200. (**Setup 2**)

| Learning Rate | One at a time | Many at a time |
|---|---|---|
| .1 | 6 | 102 |
| .2 | 147 | 104 |
| .3 | 149 | 91 |
| .4 | 149 | 116 |
| .5 | 141 | 105 |
| .6 | 157 | 114 |
| .7 | 136 | 91 |
| .8 | 137 | 91 |
| .9 | 140 | 105 |
| 1.0 | 141 | 93 |

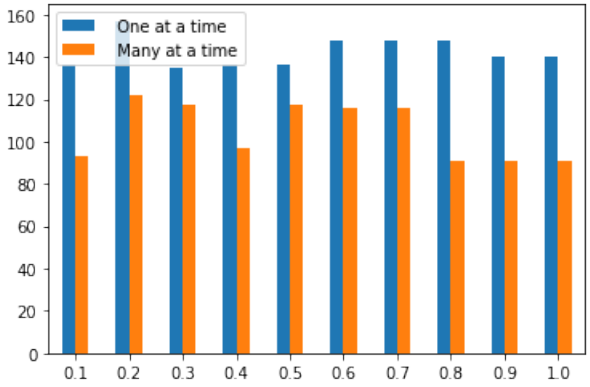
**Bar chart:**




**Weight coefficients has been set randomly by default.** Number of iterations will go up to 200. (**Setup 3**)

| Learning Rate | One at a time | Many at a time |
|---|---|---|
| .1 | 141 | 105 |
| .2 | 141 | 105 |
| .3 | 141 | 92 |
| .4 | 141 | 105 |
| .5 | 141 | 92 |
| .6 | 141 | 92 |
| .7 | 141 | 92 |
| .8 | 141 | 92 |
| .9 | 141 | 92 |
| 1.0 | 141 | 92 |

**Bar chart :**




## IV. Conclusion

So, by these experiments we can come to the conclusion that by variations of the different kind, we can optimize the performance of the classifier.

## V. Algorithm Implementation / Code

**Task 1:**

```
trainData = {}
entry = []

with open("train-perceptron.txt") as f:
    for line in f:
        entry = line.split(" ")
        if (entry[2][-1] == '\n'):
            classInfo =  entry[2][:-1]
```

```
        else:
            classInfo =  entry[2]
        trainData[(float(entry[0]),float(entry[1]))]
     = float(classInfo)
    print(trainData)


trainData1x = []
trainData1y = []
trainData2x = []
trainData2y = []

for coor , classinfo in trainData.items():
    if(classinfo == 1):
        trainData1x.append(coor[0]);
        trainData1y.append(coor[1]);
    else:
        trainData2x.append(coor[0]);
        trainData2y.append(coor[1]);

print(trainData1x , trainData1y)
print(trainData2x , trainData2y)




plt.scatter(trainData1x, trainData1y, color = 'r',
    marker = ".", label = 'Train class 1')

plt.scatter(trainData2x, trainData2y, color = 'k',
    marker = "*", label = 'Train class 2')

plt.xlabel('x - axis')
plt.ylabel('y - axis')

plt.legend(loc = 'upper center')

plt.show()
```

### Task 2:

```
highDimTrainData = {}

for coor , classinfo in trainData.items():
    if(classinfo == 1):
        highDimTrainData[(coor[0]*coor[0] , coor[1]*
    coor[1], coor[0]*coor[1], coor[0], coor[1], 1 )]
     = classinfo
    else:
        highDimTrainData[(-coor[0]*coor[0] , -coor
    [1]*coor[1], -coor[0]*coor[1], -coor[0], -coor
    [1], -1 )] = classinfo


highDimTrainData
```

### Task 3 and 4:

```
def findMisClssified(WeightVector , yValues):
    if (np.dot(WeightVector , yValues) > 0 ):
        return False
    else:
        return True


def updateWeights(WeightVector ,yValues ,
    misClassifiedIndices ,LearningRate, batchFlag):

    if(batchFlag):
        sumOfMisClassifiedIndices = np.sum(yValues[
    misClassifiedIndices], axis = 0)
    else:
        sumOfMisClassifiedIndices = yValues[
    misClassifiedIndices]

    updateFactor = sumOfMisClassifiedIndices*
    LearningRate

    WeightVector = np.sum([WeightVector ,
    updateFactor ], axis = 0)

    print(sumOfMisClassifiedIndices)
    print(updateFactor)


    return WeightVector;



def runPerceptron(highDimTrainData,WeightVector,
    NumberOfIterations,LearningRate,batchFlag):
    yValues =  np.array(list(highDimTrainData.keys()
    ))

    while( NumberOfIterations>0):
        misClassifiedIndices = []
        print("")
        interation = 200 - NumberOfIterations + 1
        print("iteration: " , interation)

        for i  in np.arange(len(yValues)):

            if(findMisClssified(WeightVector ,
    yValues[i])):
                misClassifiedIndices.append(i)
                print("misclassified at :" , i )

                if(not batchFlag):
                    WeightVector = updateWeights(
    WeightVector,yValues ,np.array([i]),LearningRate
    ,batchFlag)

        if(batchFlag):
            WeightVector = updateWeights(
    WeightVector,yValues ,misClassifiedIndices,
    LearningRate,batchFlag)
        if(len(misClassifiedIndices)<1):

            break
        NumberOfIterations-=1


    return interation



def runSetup(batchFlag ,stepSize,LearningRate ,
    WeightVector, highDimTrainData ,
    NumberOfIterations):
    numberOfInterationsHappened = []
    while(LearningRate < 1.0):
        #WeightVector = np.array([1] *
    numberOfInputs)
        interation = runPerceptron(highDimTrainData,
     WeightVector, NumberOfIterations, LearningRate,
     batchFlag)
        LearningRate += stepSize
        print(interation)
        numberOfInterationsHappened.append(
    interation)

    return numberOfInterationsHappened
```

The functon called **findMisClassified** identifies a certain example to be correctly classified or not and returns a binary value indicating it. **Updateweights** function updates the

weights. The code works for both single and batch update through flags. And finally **runPerceptron** executes the main algorithm of perceptron.