Assignment # 2: Statistical Methods in Artificial Intelligence (SMAI)

Nikhil Ranjan
Roll # - 20163027

Problem 1: Perceptron Learning

1. **Plot using the data given in Table 1 (C1 and C2)**
   Red Color: Class C1
   Blue color: Class C2

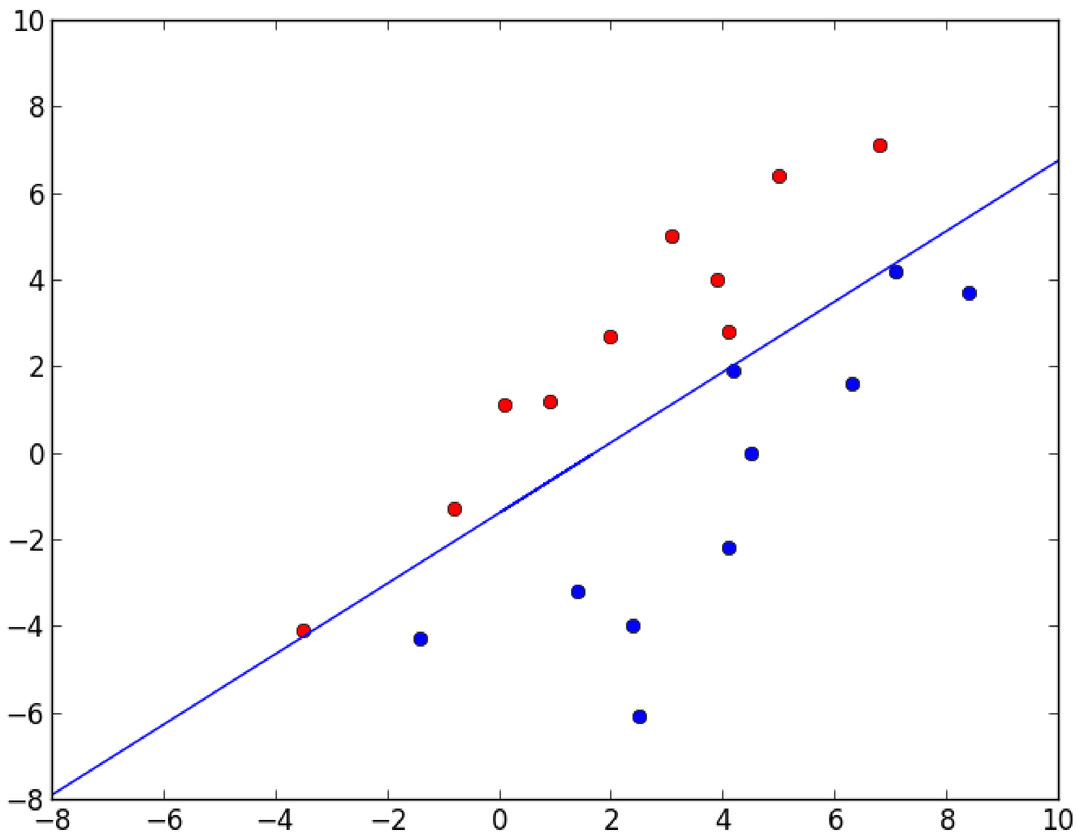

Figure: - Class C1 points with red color and Class C2 points with blue color

2. Plot using the data given in Table 2 (C1 and C2)

Red Color: Class C3

Blue color: Class C2

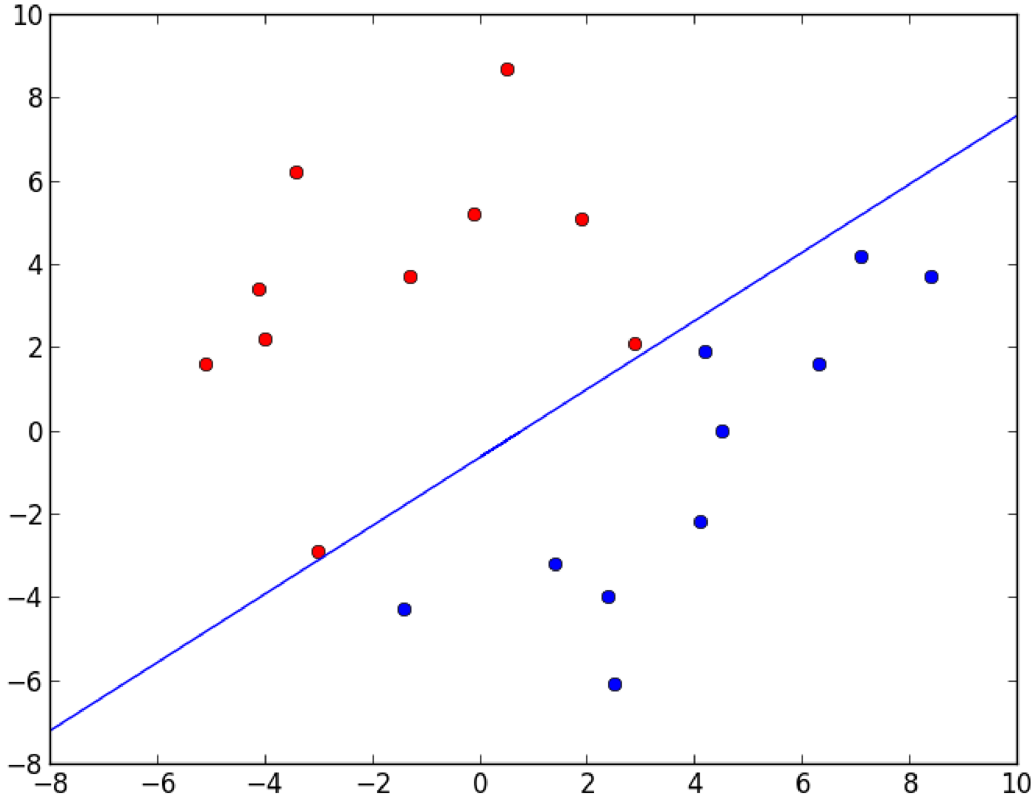

Figure: - Class C3 points with red color and Class C2 points with blue color

Sample Code: Language Python

```python
import sys
import matplotlib.pyplot as plt
import pylab as pl
import numpy as np
from matplotlib.lines import Line2D

# Make a prediction with weights
def compute(row, weights):
    bias = weights[2]
    output = bias
    #output = (w1 * X1) + (w2 * X2) + bias
    for i in range(len(row)-1):
        output += weights[i] * row[i]
    return 1 if output > 0 else 0

def getMultiplePoints(x,y,weight,boundX1,boundX2):
    x1 =[x,0]
    x2 =[0,y]
    pointsX = []
    pointsY = []
    pointsX.insert(1,y)
    pointsX.insert(2,0)
    pointsY.insert(1,0)
    pointsY.insert(2,x)
    #for boundX1
    pointsX.insert(0,boundX1)
    temp = -(weight[0]*boundX1 + weight[2])/weight[1]
    pointsY.insert(0,temp)
    #for boundX2
    pointsX.insert(3,boundX2)
    temp = -((weight[0]*boundX2) + weight[2])/weight[1]
    pointsY.insert(3,temp)
    return (pointsX,pointsY)
```

```python
#plot points
def plotCoordinates(dataset,weightPlot):
    XList1 =[]
    YList1 =[]
    XList2 =[]
    YList2 =[]
    count = 0
    boundX = -8
    boundY = 10
    #compute classifier co-ordinates
    x1 = - (weightPlot[2]/weightPlot[1])
    y1 = 0
    x2 = 0
    y2 = - (weightPlot[2]/weightPlot[0])

    # compute some random point with slope as W and bias b
    plotTup = getMultiplePoints(x1,y2,weightPlot,boundX,boundY)
    for row in dataset:
        if(count<=9):
            XList1.append(row[0])
            YList1.append(row[1])
        else:
            XList2.append(row[0])
            YList2.append(row[1])
        count = count+1
    #Draw points with red and Blue color
    plt.plot(XList1, YList1, 'ro',XList2, YList2, 'bo')
    plt.axis([boundX, boundY, boundX, boundY])
    plt.plot(plotTup[0],plotTup[1])
    plt.show()
```

```python
#Update weight and bias
def updateWeight(weights,x,l_rate,error):
    #update bias
    weights[2] = weights[2] + x[2] + l_rate * error
    #update weight part w1, w2
    for i in range(len(x)-1):
        weights[i] = weights[i] + l_rate * error * x[i]
    return weights


def findPerceptronClassifier(dataset,weights):
    flag = True
    epoch = 0
    retList = []
    l_rate = 0.2
    count = 0
    #lastWeight = []
    while(flag):
        #flag = False
        epoch = epoch + 1

        count = 0
        for row in dataset:

            predicted_val = compute(row, weights)
            error = row[-1] - predicted_val
            #update weights
            if error != 0:
                weights = updateWeight(weights,row,l_rate,error)
                count = count + 1
            lastWeight = weights
        if error == 0 and count == 0:
                flag = False
        else:
            flag = True
    retList.append(epoch)
    retList.append(weights)
    return retList
```

```python
# Input dataset for classifier
datasetC1C2 =[[0.1,1.1,0], [6.8 ,7.1,0], [-3.5 ,-4.1,0], [2.0 ,2.7,0] , [4.1 ,2.8,0] ,
        [3.1 ,5.0,0], [-0.8 ,-1.3,0],[0.9 ,1.2,0], [5.0 ,6.4,0], [3.9, 4.0,0],
        [7.1 ,4.2,1], [-1.4, -4.3,1],[4.5 ,0.0,1 ], [6.3 ,1.6,1 ],[4.2 ,1.9,1 ],
        [1.4 ,-3.2,1], [2.4 ,-4.0,1 ],[2.5 ,-6.1,1 ],[8.4 ,3.7,1], [4.1 ,-2.2,1]]


datasetC2C3 = [[-3.0 , -2.9,0], [0.5,  8.7,0], [2.9 , 2.1,0], [-0.1,  5.2,0], [-4.0 , 2.2,0], [-1.3,  3.7,0],
    [-3.4,  6.2,0], [-4.1,  3.4,0], [-5.1,  1.6,0], [1.9 , 5.1,0],[7.1 ,4.2,1],
    [-1.4, -4.3,1],[4.5 ,0.0,1 ], [6.3 ,1.6,1 ],[4.2 ,1.9,1 ],
    [1.4 ,-3.2,1], [2.4 ,-4.0,1 ],[2.5 ,-6.1,1 ],[8.4 ,3.7,1], [4.1 ,-2.2,1]]

#initialize inital weight and bias
initial_weights = [0,0,0]
#Iteration count
epoch = 0
outList = []
def C1C2Classifier():
    #Iteration count for convergence - Dataset C1 and C2
    outList = findPerceptronClassifier(datasetC1C2,initial_weights)
    epoch = outList[0]
    weightPlot = outList[1]
    print "Number of iterations required for convergence (class C1 and C2): ",epoch
    plotCoordinates(datasetC1C2,weightPlot)

def C2C3Classifier():
        #Iteration count for convergence - Dataset C2 and C3
    outList = findPerceptronClassifier(datasetC2C3,initial_weights)
    epoch = outList[0]
    weightPlot = outList[1]
    print "Number of iterations required for convergence (class C1 and C2): ",epoch
    plotCoordinates(datasetC2C3,weightPlot)

# map the inputs to the function blocks
options = {
        1 : C1C2Classifier,
            2 : C2C3Classifier,
    }
```

```python
#start
if __name__ == '__main__':
    print "1. Run C1C2 classifier \n2. Run C2C3 classifier\n"
    print "Enter your choice:\t"
    num = int(raw_input())
    options[num]()
```

Program execution and Sample Output: -

[zytham@s158519-vm Assign]$ python Q1.py
1. Run C1C2 classifier
2. Run C2C3 classifier

Enter your choice:
1
Number of iterations required for convergence (class C1 and C2): 28

[zytham@s158519-vm Assign]$ python Q1.py
1. Run C1C2 classifier
2. Run C2C3 classifier

Enter your choice:
2
Number of iterations required for convergence (class C1 and C2): 13

3. Difference on the number of iteration required for convergence in above two cases

Number of iterations required for convergence (class C1 and C2): 28

Number of iterations required for convergence (class C1 and C2): 13

Problem 2: Voted Perceptron

1. Write the program to implement Voted Perceptron.
   Sample code in python attached end of this question.

2. Report – Accuracy Vs Epoch count for both dataset - Breast Cancer Dataset and Ionosphere Dataset

Breast Cancer Dataset - Epoch count on X axis and Accuracy on Y axis

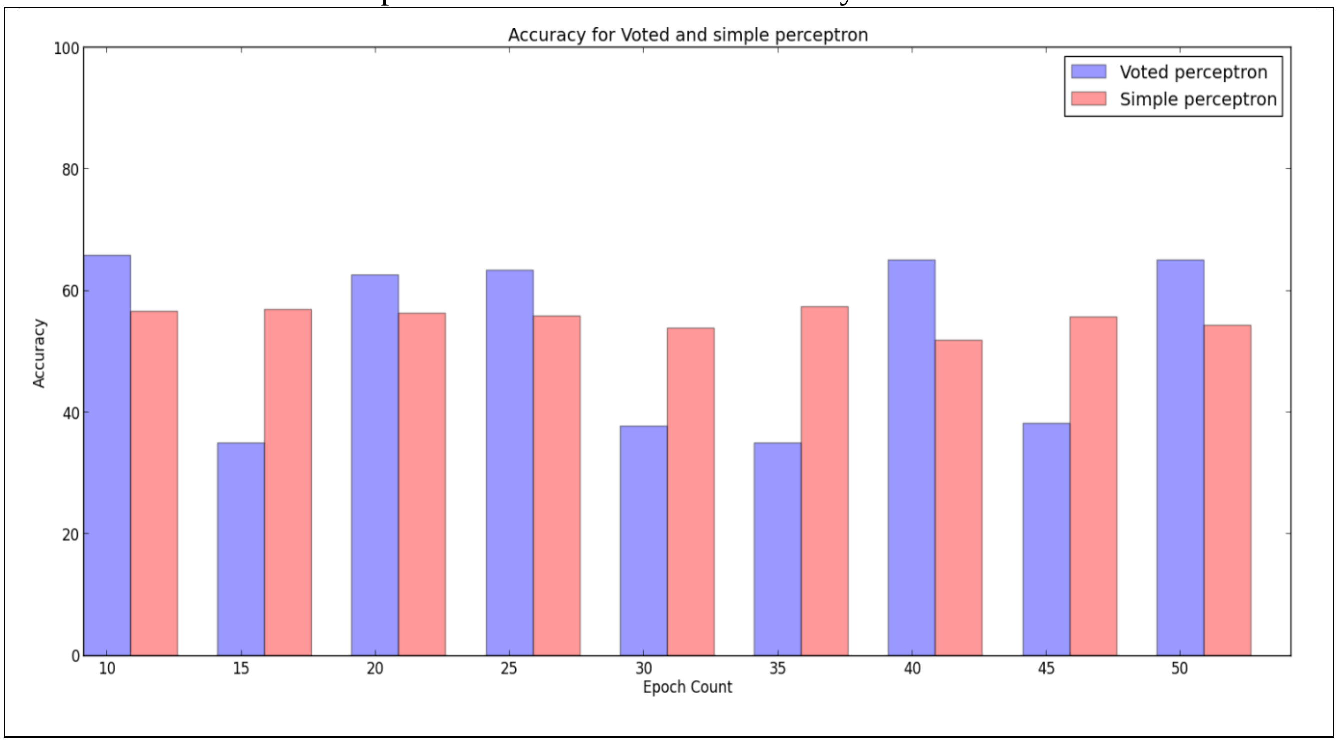

Figure: - Epoch count with accuracy of Voted and Simple perceptron (Blue color: Voted perceptron and Pink Color: Simple perceptron)

Ionosphere Dataset - Epoch count on X axis and Accuracy on Y axis

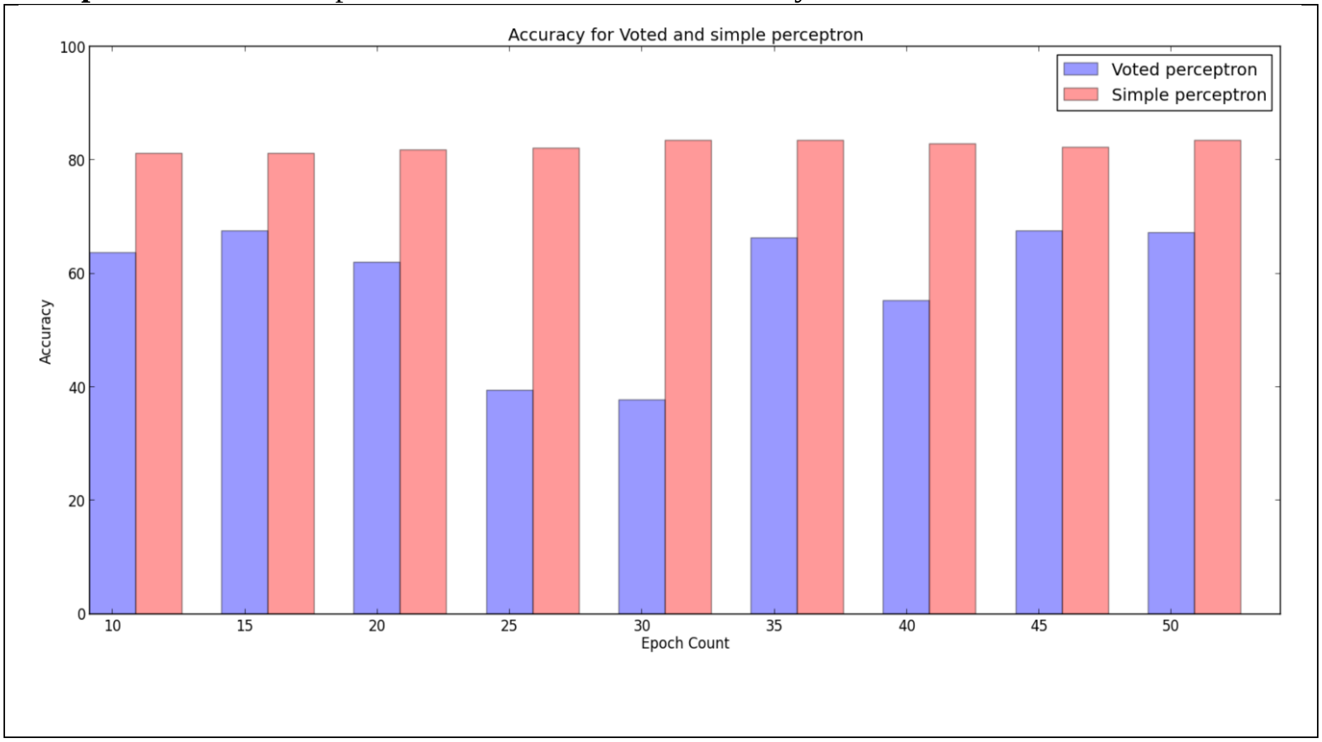

Figure: - Epoch count with accuracy of Voted and Simple perceptron (Blue color: Voted perceptron and Pink Color: Simple perceptron)

3. **Plot the classifiers learnt at the end of Epoch number** – Plotting cannot be done, providing learned weight vector by simple perceptron.

Weight vector by simple perceptron – Breast Cancer Dataset

Epoch	Weight vector
10	[-46.0000000000006, 69.19999999999747, 31.200000000000717, 5.400000000000016, -86.0000000000016, 41.999999999999226, -62.80000000000072, 25.200000000000983, -34.19999999999918, 250.00000000000733]
15	[-54.4000000000018, 97.79999999999515, 37.60000000000098, 15.20000000000038, -127.40000000000748, 71.39999999999614, -71.80000000000257, 36.20000000000029, -60.40000000000322, 378.4000000000171]
20	[-79.60000000000299, 140.59999999999738, 20.4000000000011, 13.40000000000107, -

	143.8000000000091, 85.39999999999435, -121.60000000000606, 45.79999999998064, -83.4000000000586, 502.6000000000255]
25	[-104.80000000000373, 172.20000000000604, 45.1999999999895, 23.400000000001658, -222.6000000001375, 104.1999999999185, -140.6000000000041, 71.99999999999254, -76.2000000000725, 651.5999999999844]
30	[-119.80000000000439, 163.2000000000031, 91.19999999999204, 31.800000000002886, -247.6000000001605, 145.1999999999954, -164.39999999999927, 60.999999999994266, -98.00000000001053, 772.5999999999543]
35	[-146.39999999999964, 222.40000000001967, 72.3999999999283, 25.200000000003463, -284.2000000000034, 143.59999999999636, -203.79999999999188, 97.1999999998706, -115.6000000000137, 929.1999999999098]
40	[-153.59999999999775, 256.2000000000286, 65.99999999999191, 1.3999999999997403, -309.5999999999904, 180.20000000001215, -216.1999999998913, 90.59999999998841, -150.8000000000009, 977.1999999998894]
45	[-176.5999999999901, 302.0000000000436, 81.7999999998817, 45.99999999999779, -363.59999999995864, 164.20000000000783, -263.9999999998124, 127.1999999999823, -161.39999999999557, 1161.3999999998293]
50	[-193.5999999999852, 315.00000000005315, 88.999999999985, 12.000000000001435, -380.7999999999489, 241.20000000003657, -294.7999999999756, 107.39999999997988, -221.79999999996463, 1274.7999999998037]

Weight vector by simple perceptron – Ionosphere Dataset

Epoch	Weight Vector
10	[-54.600000000000236, 0.0, -20.51315400000006, -3.091660000000004, -17.678098000000052, -10.09598200000001, -14.72310200000001, -11.218933999999981, -7.737659999999996, -12.879687999999978, -4.121087999999991, -3.9294019999999943, 1.0690479999999993, -5.178093999999999, -0.6236600000000007, 2.6609439999999984, -1.174049999999999, -4.346694000000005, -0.02372000000000063, 0.6659980000000006, -5.4893299999999945, 10.28277799999999, -2.926405999999992, -1.0291119999999998, -4.663545999999989, -3.2120420000000016, -0.7655440000000002, 3.058198000000004, -2.1674960000000065, -3.8246999999999947, -7.760083999999994, 3.372080000000003, -0.16120200000000096, 5.077731999999995, 113.79999999999949]
15	[-81.40000000000062, 0.0, -30.38785600000001, -3.565068000000003, -25.797796000000073, -16.781713999999972, -18.782650000000043, -18.874885999999994, -12.451140000000004, -17.74081199999997, -6.354503999999966, -5.103959999999991, -1.3682060000000045, -6.590603999999993, -1.772381999999999, 2.855213999999997, -2.243699999999999, -4.955234000000007, -1.213652000000004, 0.7082460000000014, -5.701159999999989, 14.812011999999967, -2.8402639999999897, -0.22665000000000052, -3.6090839999999877, -1.5888160000000007, -0.32018399999999947, 5.844960000000005, -3.1388759999999993, -8.069346000000014, -8.366651999999995, 2.843802000000002, 0.4319380000000003, 6.135307999999986, 164.60000000000045]
20	[-102.80000000000092, 0.0, -33.05405200000006, -3.0888039999999912, -29.523366000000177, -20.02224399999997, -21.731526000000027, -22.113313999999907, -14.725942000000046, -21.19513200000003, -6.363889999999951, -3.6340039999999854, -1.8429700000000089, -4.335272, -1.6875179999999992, 1.7221219999999982, -4.606507999999991, -5.997835999999992, 0.5262999999999942, 3.7137839999999986, -4.877863999999988, 17.635297999999946, -0.8860880000000108, -1.8257960000000044, -3.814628000000002, -1.7212119999999984,

	-2.5980780000000006, 7.69885800000003, -4.801043999999994, -7.933494000000024, -12.007538000000082, 3.804850000000001, 3.681028000000018, 6.882231999999983, 192.6000000000013]
25	[-135.60000000000085, 0.0, -39.995667999999895, -4.302923999999998, -39.05787799999992, -23.266513999999983, -29.148912000000017, -33.99129199999989, -18.841224000000015, -28.157682000000076, -7.1183559999999, -2.880885999999988, -0.7142220000000032, -2.018339999999989, -3.0927200000000052, 2.694305999999995, -6.952657999999978, -7.6601720000000055, -0.6748520000000053, 4.61296199999999, -5.3821079999999855, 24.009741999999942, -2.2479539999999867, -1.8236900000000027, -7.205619999999957, -4.2468860000000115, -1.676142000000008, 8.691522000000019, -3.108016000000006, -12.114876000000026, -13.269960000000102, 2.7691140000000054, 3.3630840000000397, 12.143121999999996, 252.40000000000316]
30	[-157.3999999999996, 0.0, -51.67720399999958, -7.683734000000037, -48.42121599999978, -27.989482000000002, -33.200025999999944, -37.338803999999925, -19.464472000000008, -30.992012000000102, -12.227088000000053, -3.6351119999999897, -1.1619780000000066, -8.917619999999994, -3.0234040000000015, 1.2689419999999962, -5.7044979999999885, -9.243270000000006, 0.04078599999999656, 5.094914000000001, -7.320789999999998, 24.80472399999983, -2.7799359999999718, -0.7109240000000018, -3.9854659999999806, -1.4715819999999955, 0.181106000000001, 8.712472000000012, -7.133577999999993, -11.932676000000017, -18.27234800000014, 3.2385360000000065, 1.651025999999994, 8.961345999999992, 300.60000000000446]
35	[-144.40000000000035, 0.0, -42.917965999999886, -3.3471999999999986, -36.315292000000014, -25.887697999999933, -25.759118000000125, -30.171395999999863, -16.288534000000062, -28.051020000000047, -6.226647999999916, 1.0817860000000012, -1.3353300000000035, -7.949719999999982, 0.18935800000000014, 1.100217999999996, -11.290928000000036, -8.758317999999989, 1.403263999999995, 1.781218000000006, -2.3260379999999956, 25.839281999999905, 2.9455620000000047, 7.536898000000059, -7.506909999999959, -5.0525320000000224, -0.14443800000000068, 8.556347999999991, -5.783521999999985, -

	13.528100000000027, -15.177404000000152, 2.756604, -0.6569100000000047, 10.190283999999993, 251.6000000000032]
40	[-202.19999999999706, 0.0, -62.57703599999946, -7.250426000000073, -56.17124799999964, -32.21655600000006, -38.17911599999987, -46.88547199999992, -25.106207999999928, -42.92586799999997, -13.335396000000115, -1.2001819999999739, -0.5590800000000082, -9.716383999999987, -5.020406000000008, 3.716497999999988, -6.975001999999994, -13.812581999999944, -2.5236359999999967, 3.553752000000005, -10.056034000000079, 35.88137399999982, -5.353085999999942, 5.693500000000033, -6.013847999999914, -5.657364000000042, -2.4119180000000098, 13.216668, -6.884097999999978, -19.939372000000027, -18.888110000000193, 7.319671999999964, 0.31702599999999836, 13.554425999999976, 370.60000000000684]
45	[-217.79999999999617, 0.0, -71.21843599999904, -15.363654000000057, -62.43296199999919, -43.208378000000046, -46.351061999999665, -48.0121500000001, -32.293979999999834, -36.944234000000016, -13.969452000000143, -7.223777999999974, -3.751205999999982, -7.1316759999999295, -10.44038400000008, 9.470324000000035, -10.19572400000004, -16.218223999999942, -4.815405999999923, 6.724700000000005, -11.382042000000125, 35.313971999999815, -0.7639299999999871, -1.2665200000000205, -12.411166000000089, -0.7386899999999956, -1.0827940000000043, 13.008276000000082, -6.90662599999996, -24.95680199999997, -24.041822000000053, 7.552357999999961, 12.37279400000003, 17.02003800000008, 425.40000000000816]
50	[-240.79999999999487, 0.0, -71.61208199999912, -10.389112000000056, -67.43171399999946, -46.03299800000026, -51.103507999999614, -54.26086200000008, -32.18206199999996, -45.291413999999804, -12.818772000000106, -4.617433999999979, -6.331617999999995, -11.883459999999994, -3.247994000000009, 0.4554239999999924, -15.665542000000109, -21.11079399999977, -1.6478940000000122, 5.616760000000012, -10.076766000000054, 42.050877999999734, -0.9767280000000115, 7.2220640000001115, -9.01391799999996, -7.666552000000056, -4.000575999999993, 12.613984000000025, -10.826746000000027, -18.804673999999995, -28.60105600000017, 7.481465999999982, 1.5870279999999757, 10.04061, 437.0000000000092]

Sample code: - Language Python

```python
import sys
import matplotlib.pyplot as plt
import numpy as np
from copy import deepcopy
import random
import math

# Make a prediction with weights - Voted perceptron
def predictVotedPerceptron(row, weights,bias):
    row_size = len(row)
    #Outcome of this row - Yi
    y = row[row_size-1]
    output = bias
    for i in range(len(weights)):
        output += weights[i] * row[i]
    output = output*y
    return 1.0 if output > 0.0 else -1.0

#Update weight and bias - for simple perceptron
def updateWeight_bias(weights,x,l_rate,error):
    lenW = len(weights)
    lenR = len(x)
    #update bias
    weights[lenW-1] = weights[lenW-1] + x[lenR-1] + l_rate * error
    #update weight part w1, w2
    for i in range(len(x)-1):
        weights[i] = weights[i] + l_rate * error * x[i]
    return weights

#Make a prediction with weights - Perceptron simple
def predictSP(row, weights):
    lenW = len(weights)
    outcome = weights[lenW-1] #biasat last index
    for i in range(lenW-1): # minus 1 because of last index is bias
        outcome += weights[i] * row[i]
    return 1.0 if outcome >= 0.0 else 0.0
```

```python
def updateDataSetWithLabelSimplePerceptron(dataset,labelClass,file1Or2): #change label 1 = 0,label 2 = 1
    if len(labelClass) == 2: #Assume two class labels coming only
        label1 = labelClass[0]
        label2 = labelClass[1]
    datasetDup = []
    indexDup = 0
    if file1Or2 == 1:
        rowIndex = 1
    elif file1Or2 == 2:
        rowIndex = 0
    rowSize = len(dataset[0])
    for row in dataset:
        if row[rowSize -1] == label1:
            row[rowSize -1] = 0.0
        elif row[rowSize -1] == label2:
            row[rowSize -1] = 1.0
        datasetDup.insert(indexDup,deepcopy(row[rowIndex:])) # keep on adding element at start of index
    return datasetDup

def updateDataSetWithLabelVotedPerceptron(dataset,labelClass,file1Or2): #change label 1 = -1,label 2 = 1
    if len(labelClass) == 2: #Assume two class labels coming only
        label1 = labelClass[0]
        label2 = labelClass[1]
    datasetDup = []
    indexDup = 0
    if file1Or2 == 1:
        rowIndex = 1
    elif file1Or2 == 2:
        rowIndex = 0
    rowSize = len(dataset[0])
    for row in dataset:
        if row[rowSize -1] == label1:
            row[rowSize -1] = -1.0
        elif row[rowSize -1] == label2:
            row[rowSize -1] = 1.0
        datasetDup.insert(indexDup,deepcopy(row[rowIndex:]))
    return datasetDup
```

```python
def trainPerceptronWeights(dataset,featuresCount,epochCount,l_rate):
    weights = [0.0 for i in range(featuresCount+1)] # +1 for adding bias at last index
    count = 0
    for epoch in range(epochCount):
        count = 0
        for row in dataset:
            predicted_val = predictSP(row, weights)
            error = row[-1] - predicted_val
            if error != 0:
                weights = updateWeight_bias(weights,row,l_rate,error)
    return weights

#Update weight and bias
def updateWeightBias(weight,row, bias):
    row_size = len(row)
    listOut = []
    #Outcome of this row - Yi
    y = row[row_size-1]
    #update bias
    bias = bias + y
    #update weight part
    for i in range(len(weight)):
        weight[i] = weight[i] + row[i]*y
    listOut.append(weight)
    listOut.append(bias)
    return listOut

def createWeightVector(weight,bias,c):
    weightVector = []
    weightVector.insert(0,weight)
    weightVector.insert(1,bias)
    weightVector.insert(2,c)
    return weightVector
```

```python
def votedPerceptronTraining(dataset,epochCount,featureLen):
    n = 1;b = 0;c = 1;bias = 0
    m = len(dataset)
    outcome = []
    indexOut = 0
    #make weight vector
    weight = [0 for x in range(featureLen)]
    weightVector = createWeightVector(weight,bias,c)
    outcome.insert(indexOut,deepcopy(weightVector))
    indexOut = indexOut +1
    for iter in range(epochCount):
        for row in dataset:
            predict_val = predictVotedPerceptron(row,weight,bias)
            error = row[-1] - predict_val
            if error != 0.0:
                listOut = updateWeightBias(weight,row,bias)
                #update Outcome list
                weightVector = createWeightVector(listOut[0],listOut[1],c)
                weight = listOut[0]
                bias = listOut[1]
                outcome.insert(indexOut,deepcopy(weightVector))
                indexOut = indexOut + 1
                n = n+1
                c = 1
            else:
                c = c +1
    return outcome

def predictOutcome(outcomeWeightVector,row):
    row_size = len(row)
    signList = []
    #Outcome of this row - Y is actual output
    y = row[row_size-1]
    for weightRowBias in outcomeWeightVector:
        bias = weightRowBias[1] #bias at [2]
        c = weightRowBias[2]
        weights = weightRowBias[0]
        output = bias
```

```
            output += weights[i] * row[i]
        if output <= 0:
            output = -1
        else:
            output = 1
        output = output*c
        signList.append(output)
    Yprime = sum(signList)
    if Yprime <= 0:
        Yprime = -1
    else:
        Yprime = 1
    #compare sign of Yprime and Y
    if Yprime == y:
        return True
    else:
        return False

def computeAccuracyVotedPerceptron(outcomeWeightVector,testData):
    countTotal = len(testData)
    correct = 0
    wrong = 0
    for row in testData:
        value = predictOutcome(outcomeWeightVector,row)
        if value :
            correct = correct  +1
        else :
            wrong = wrong +1
    return (correct,wrong)

def computeAccuracySimplePerceptron(weights,testData):
    countTotal = len(testData)
    correct = 0
    wrong = 0
    rowSize = len(testData[0])
    for row in testData:
        value = predictSP(row, weights)
        if value == row[rowSize-1] :
            correct = correct  +1
```

```
        else :
            wrong = wrong +1
    return (correct,wrong)


#read file and convert input into list
def read_file_content(filename,file1Or2):
    file_handle = open(filename)
    allLines = file_handle.readlines()
    index = 0
    inputList = []
    for eachline in allLines:
        eachline = eachline.rstrip('\n')
        data = eachline.strip().split(',')
        intList = []
        dataLen = len(data)
        if file1Or2 == 1:
            for each in data:
                intList.append(int(each))
        elif file1Or2 == 2:
            index = 0
            for each in data:
                if index < dataLen-1:
                    intList.append(float(each))
                    index = index + 1
                else:
                    intList.append(each)
            inputList.insert(index,intList)
    return inputList

def datasetup(fileName,file1Or2):
    inputList = read_file_content(fileName,file1Or2)
    return inputList

def computeMeanAccuracyVotedPerceptron(dataset,epochCount,num_folds=10):
    total = len(dataset)
    featureCount = len(dataset[0]) -1  #featuresCount = 9 for  cancer data
    accuracy = 0.0
    subset_size = len(dataset)/num_folds
```

```python
        random.shuffle(dataset)
    for i in range(num_folds):
        testDatalist = dataset[i*subset_size:][:subset_size]
        trainingList = dataset[:i*subset_size] + dataset[(i+1)*subset_size:]
        outcomeWeightVector = votedPerceptronTraining(trainingList,epochCount,featureCount)
        (correctPrecdictCount,wrongPrecdictCount) =
                computeAccuracyVotedPerceptron(outcomeWeightVector,testDatalist)
        t = correctPrecdictCount + wrongPrecdictCount
        accuracy = accuracy + float(correctPrecdictCount) / t
    accuracyMean = (float(accuracy)*100)/num_folds
    return accuracyMean

def computeMeanAccuracySimplePerceptron(dataset,epochCount,num_folds=10): #num_folds = k
    l_rate = 0.2
    total = len(dataset)
    featureCount = len(dataset[0]) - 1
    subset_size = len(dataset)/num_folds
    accuracy = 0.0
    random.shuffle(dataset)
    for i in range(num_folds):
        testDatalist = dataset[i*subset_size:][:subset_size]
        trainingList = dataset[:i*subset_size] + dataset[(i+1)*subset_size:]
        trainiwdWeights = trainPerceptronWeights(trainingList,featureCount,epochCount,l_rate)
        (correctPrecdictCount,wrongPrecdictCount) =
                computeAccuracySimplePerceptron(trainiwdWeights,testDatalist)
        t = correctPrecdictCount + wrongPrecdictCount
        accuracy = accuracy + (float(correctPrecdictCount) / t)

    accuracyMean = (float(accuracy)*100)/num_folds
    return (accuracyMean,trainiwdWeights)


def plotAccuracyWithEpoch(accuracyWithEpochVP,accuracyWithEpochSP):
    accListVP= []
    labelListVP =[]
    keylistVP = accuracyWithEpochVP.keys()
    keylistVP.sort()
    for key in keylistVP:
        accListVP.append(accuracyWithEpochVP[key])
```

```
    labelListVP.append(str(key))
accListSP= []
keylistSP = accuracyWithEpochSP.keys()
keylistSP.sort()
for key in keylistSP:
    accListSP.append(accuracyWithEpochSP[key])
n_epoch = len(accuracyWithEpochVP)
voted_acc= tuple(accListVP)
simpleperc_acc = tuple(accListSP)

fig, ax = plt.subplots()
ax.set_ylim([0, 100])
index = np.arange(n_epoch)
bar_width = 0.35

opacity = 0.4
error_config = {'ecolor': '0.3'}

rects1 = plt.bar(index, voted_acc, bar_width,
                 alpha=opacity,
                 color='b',
                 error_kw=error_config,
                 label='Voted perceptron')

rects2 = plt.bar(index + bar_width, simpleperc_acc, bar_width,
                 alpha=opacity,
                 color='r',
                 error_kw=error_config,
                 label='Simple perceptron')

plt.xlabel('Epoch Count')
plt.ylabel('Accuracy')
plt.title('Accuracy for Voted and simple perceptron ')

#plt.xticks(index + bar_width / 2, ('10','15', '20', '25', '30', '35','40', '45', '
plt.xticks(index + bar_width / 2, tuple(labelListVP))
plt.legend()
plt.tight_layout()
plt.show()
```

```
def DatasetCancer():
    filename1 = "breast-cancer-wisconsin.data.txt"
    labelClass = [2,4]
    k_fold = 10
    epochList = [10,15, 20, 25, 30, 35,40, 45, 50]
    accuracyWithEpochVP = {}
    accuracyWithEpochSP = {}
    #setup data from file
    datasetBSW = datasetup(filename1,1)

    #create a copy of dataset for Voted Perceptron
    datasetVP = deepcopy(datasetBSW)

    #Update label with appropriate value
    datasetSP= updateDataSetWithLabelSimplePerceptron(datasetBSW,labelClass,1)
    datasetVP= updateDataSetWithLabelVotedPerceptron(datasetVP,labelClass,1)

    for epochCount in epochList:
        accuracyMean =
            computeMeanAccuracyVotedPerceptron(datasetVP,epochCount,k_fold)
        accuracyWithEpochVP[epochCount] = accuracyMean

    epochWeightvectorMap = {}
    #for simple perceptron
    for epochCount in epochList:
        #k= 10, report 10-fold cross validation accuracies
        (accuracyMean,trainiwdWeights) =
                computeMeanAccuracySimplePerceptron(datasetSP,epochCount,k_fold)
        accuracyWithEpochSP[epochCount] = accuracyMean
        epochWeightvectorMap[epochCount] = trainiwdWeights

    #weight vector learned with epoch
    keylistSP = epochWeightvectorMap.keys()
    keylistSP.sort()
    for key in keylistSP:
        print "Epoch :%d : weight learned\n ",key,epochWeightvectorMap[key]
```

```python
def DatasetIomosphere():
    filename2 = "ionosphere.data.txt"
    labelClass = ['g','b']
    k_fold = 10
    epochList = [10,15, 20, 25, 30, 35,40, 45, 50]
    accuracyWithEpochVP = {}
    accuracyWithEpochSP = {}
    #setup data from file
    datasetION = datasetup(filename2,2)
    #create a copy of dataset for Voted Perceptron
    datasetVP = deepcopy(datasetION)

    #Update label with appropriate value
    datasetSP= updateDataSetWithLabelSimplePerceptron(datasetION,labelClass,2)
    datasetVP= updateDataSetWithLabelVotedPerceptron(datasetVP,labelClass,2)

    for epochCount in epochList:
        accuracyMean = computeMeanAccuracyVotedPerceptron(datasetVP,epochCount,k_fold)
        accuracyWithEpochVP[epochCount] = accuracyMean

    epochWeightvectorMap = {}
    #for simple perceptron
    for epochCount in epochList:
        #k= 10, report 10-fold cross validation accuracies
        (accuracyMean,trainiwdWeights) =
            computeMeanAccuracySimplePerceptron(datasetSP,epochCount,k_fold)
        accuracyWithEpochSP[epochCount] = accuracyMean
        epochWeightvectorMap[epochCount] = trainiwdWeights

    #weight vector learned with epoch
    keylistSP = epochWeightvectorMap.keys()
    keylistSP.sort()
    for key in keylistSP:
        print "Epoch :%d : weight learned\n ",key,epochWeightvectorMap[key]

    #plot accuracy vs Epoch count
    plotAccuracyWithEpoch(accuracyWithEpochVP,accuracyWithEpochSP)
```

```python
# map the inputs to the function blocks
options = {
        1 : DatasetCancer,
            2 : DatasetIomosphere,
    }
#start
if __name__ == '__main__':
    print "1. Epoch Vs Accuracy on dataset breast-cancer-wisconsin.data.txt \
\n2. Epoch Vs Accuracy on dataset ionosphere.data.txt \n"
    print "Enter your choice:\t"
    num = int(raw_input())
    options[num]()
```

Problem 3: Least Square Approach [5 Marks]

1. Linear classifier using least square approach
2. Linear classifier using Fisher's linear discriminant.
   Sample code in python for both classifiers shown below.

3. Display data points and both classifier on Table -1 data set.
   Least square approach - **Black color**
   Fisher's linear discriminant – **Green color**

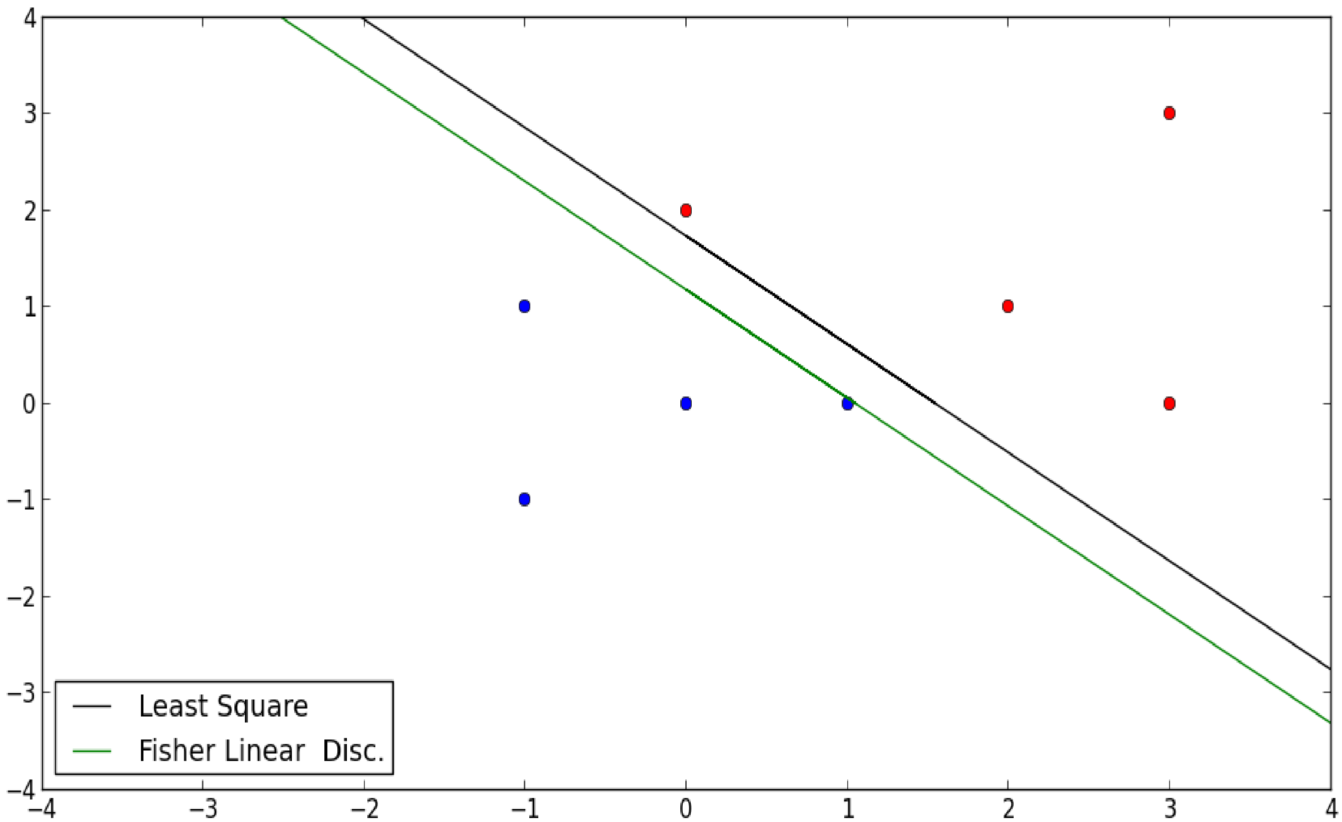

Figure (Analysis on Table -1 data set.): - Least Square classifier in **black** and Fisher Classifier in **Green**

4. Display data points and both classifier on Table -2 data set.
   Least square approach - **Black** color
   Fisher's linear discriminant – **Green color**

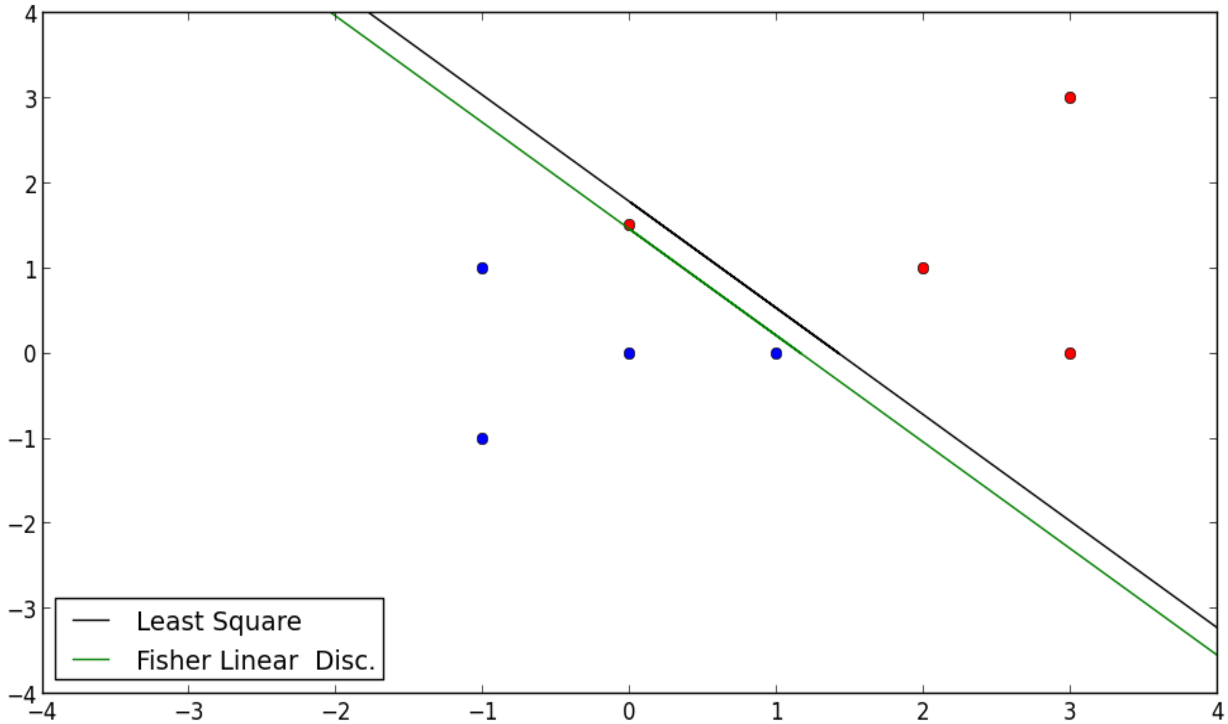

Figure (Analysis on Table -2 data set.): -   Least Square classifier in **black** and Fisher Classifier in **Green**

Sample Code: Language Python

```python
import sys
import matplotlib.pyplot as plt
import pylab as pl
import numpy as np

def getMultiplePoints(x,y,weight,boundX1,boundX2):
    x1 =[x,0]
    x2 =[0,y]
    pointsX = []
    pointsY = []
    pointsX.insert(1,y)
    pointsX.insert(2,0)
    pointsY.insert(1,0)
    pointsY.insert(2,x)
    #for boundX1
    pointsX.insert(0,boundX1)
    temp = -(weight[0]*boundX1 + weight[2])/weight[1]
    pointsY.insert(0,temp)
    #for boundX2
    pointsX.insert(3,boundX2)
    temp = -((weight[0]*boundX2) + weight[2])/weight[1]
    pointsY.insert(3,temp)
    return (pointsX,pointsY)

#plot points
def getCoordinatesList(dataset,weightPlot):
    XList1 =[]
    YList1 =[]
    XList2 =[]
    YList2 =[]
    count = 0
    boundX = -4
    boundY =  4
    #compute classifier co-ordinates
    x1 = - (weightPlot[2]/weightPlot[1])
    y1 = 0
    x2 = 0
    y2 = - (weightPlot[2]/weightPlot[0])
    itr = len(dataset)/2
```

```python
    # compute some random point with slope as W and bias b
    plotTup = getMultiplePoints(x1,y2,weightPlot,boundX,boundY)
    for row in dataset:
        if(count< itr):
            XList1.append(row[0])
            YList1.append(row[1])
        else:
            XList2.append(row[0])
            YList2.append(row[1])
        count = count+1
    return (XList1, YList1,XList2, YList2,plotTup)

def plotDataPointsAndClassifier(plotData,weightPlotLS,weightPlotFisher):
    boundX = -4
    boundY =  4
    colorLS = 'black'
    colorFLD = 'green'
    (XList1, YList1,XList2, YList2,plotTupLS) = getCoordinatesList(plotData,weightPlotLS)
    (XList11, YList11,XList21, YList21,plotTupFisher) = getCoordinatesList(plotData,weightPlotFisher)
    #Draw points with red and Blue color
    plt.plot(XList1, YList1, 'ro',XList2, YList2, 'bo')
    plt.axis([boundX, boundY, boundX, boundY])
    plt.plot(plotTupLS[0],plotTupLS[1],color = colorLS,label='Least Square')
    plt.plot(plotTupFisher[0],plotTupFisher[1],color = colorFLD, label = 'Fisher Linear  Disc.')
    plt.legend(loc='best')
    plt.show()

def compute(row, weights):
    bias = weights[2]
    output = bias
    for i in range(len(row)-1):
        output += weights[i] * row[i]
    if row[2] == 1 and output > 0:
        return True
    elif row[2] == -1 and output <= 0:
        return True
    else:
        return False
```

```python
#compute b to such data data point are segrated
def getB(dataset,weights):
    flag = True
    epoch = 1
    while(True):
        flag = False ; epoch = epoch + 1
        for row in dataset:
            prediction = compute(row, weights)
            if not prediction:
                weights[2] = row[2] -(weights[0]*row[0]+weights[1]*row[1])
                flag = True
        if epoch == 10 or flag == False :
            break
    return weights

def FisherClassifier(datasetC1,datasetC2):
    inputC1 = np.matrix(datasetC1)
    inputC2 = np.matrix(datasetC2)
    #column wise mean
    mean1 = inputC1.mean(0)
    mean2 = inputC2.mean(0)
    diffMean = mean1 - mean2
    #Tranposed class value
    inputC1T = inputC1.getT()
    inputC2T = inputC2.getT()
    #find covariance
    S1 = np.cov(inputC1T)
    S2 = np.cov(inputC2T)
    #find Sw -> within-class scatter matrix
    Sw = S1+S2
    #Inverse of Sw
    SwInv = np.matrix(Sw).getI()
    #Find weight vector
    w = np.matrix(SwInv) * diffMean.getT()
    #find One=D vector
    y1 = w.getT()* inputC1T
    y2 = w.getT()* inputC2T
    #compute weight for ploting classifier
    weightPlot = []
```

```
    weightPlot.insert(0,w.item(0))
    weightPlot.insert(1,w.item(1))
    #Randomly select value for start = 0 or -0.3 or - 0.9
    weightPlot.insert(2,-0.9)
    return weightPlot

# To find classifier Minimum Squared Error Procedures - using Pseudoinverse
def LeastSquareClassifier(inputData):
    #Compute b based on input size. B is 1x<size> matrix with 1
    size = len(inputData)
    b = [1 for x in range(size)]
    #find b's transpose - > 8x1 matrix
    bt = np.matrix(b).getT()
    #Prepare input matrix from dataset
    m = np.matrix(inputData)
    #find tranpose of input matrix
    t = m.getT()
    #Multiply transpose of input matrix and matrix -  (Y^tY)
    mul = t*m
    #find inverse of outcome of above operation - (Y^tY)^-1
    inv = mul.getI()
    # Find pseudo inverse- Multiply inversed matrix with transpose of input matrix - (Y^tY)^-1Y^t
    secondMul = inv * t
    #find solution matrix - Multiply pseudo matrix with b
    f = secondMul * bt
    #compute weight for ploting classifier
    weightPlot = []
    weightPlot.insert(0,f.item(1))
    weightPlot.insert(1,f.item(2))
    weightPlot.insert(2,f.item(0))
    return weightPlot

def ClassifierOnTable1():
    #Find least square classifier weight
    inputData = [[1,3,3], [1,3,0],[1,2,1] ,[1,0,2] ,[-1,1 ,-1],[-1,0, 0],[-1,1,1],[-1,-1 ,0]]
    plotData =  [[3,3,1], [3,0,1],[2,1,1] ,[0,2,1] ,[-1 ,1,-1],[0, 0,-1],[-1,-1,-1],[1,0,-1]]
    # find classifier for given dataset and Plot it.
    weightPlotLS = LeastSquareClassifier(inputData)
```

```
    #Find Fisher classifier weight
    datasetC1 = [[3,3], [3,0],[2,1] ,[0,2]]
    datasetC2 = [[-1,1],[0, 0],[-1,-1],[1,0]]
    #plotDataFisher =  [[3,3,1], [3,0,1],[2,1,1] ,[0,2,1] ,[-1 ,1,-1],[0, 0,-1],[-1,-1,-1],[1,0,-1]]
    # find classifier for given dataset and Plot it.
    weightPlot2 = FisherClassifier(datasetC1,datasetC2)
    #Get approximate value of b
    weightPlotFisher = getB(plotData,weightPlot2)
    #plot data points and classifier
    plotDataPointsAndClassifier(plotData,weightPlotLS,weightPlotFisher)

def ClassifierOnTable2():
    inputData = [[1,3,3], [1,3,0],[1,2,1] ,[1,0,1.5] ,[-1,1 ,-1],[-1,0, 0],[-1,1,1],[-1,-1 ,0]]
    plotData  = [[3,3,1], [3,0,1],[2,1,1] ,[0,1.5,1] ,[-1 ,1,-1],[0, 0,-1],[-1,-1,-1],[1,0,-1]]
    # find classifier for given dataset and Plot it.
    weightPlotLS = LeastSquareClassifier(inputData)
    datasetC1 = [[3,3], [3,0],[2,1] ,[0,1.5]]
    datasetC2 = [[-1,1],[0, 0],[-1,-1],[1,0]]
    plotData =  [[3,3,1], [3,0,1],[2,1,1] ,[0,1.5,1] ,[-1 ,1,-1],[0, 0,-1],[-1,-1,-1],[1,0,-1]]
    # find classifier for given dataset and Plot it.
    # find classifier for given dataset and Plot it.
    weightPlot2 = FisherClassifier(datasetC1,datasetC2)
    #Get approximate value of b
    weightPlotFisher = getB(plotData,weightPlot2)
    #plot data points and classifier
    plotDataPointsAndClassifier(plotData,weightPlotLS,weightPlotFisher)

# map the inputs to the function blocks
options = {
        1 : ClassifierOnTable1,
        2 : ClassifierOnTable2,
    }
```

```
#start
if __name__ == '__main__':
    Dataset1C1 = [ [3,3], [3,0],[2,1] ,[0,1.5]]
    Dataset1C2 = [[-1 ,1],[0, 0],[-1,-1],[1 ,0]]
    print "1. Classifier(FisherClassifier and LeastSquareClassifier) on Data points in Table 1
\n 2. Classifier(FisherClassifier and LeastSquareClassifier) on Data points in Table 2 \n"
    print "Enter your choice:\t"
    num = int(raw_input())
    options[num]()
```

<u>Program execution and Sample Output:</u> -

[zytham@s158519-vm SMAI]$ python Q3M.py
1. Classifier(FisherClassifier and LeastSquareClassifier) on Data points in Table 1
2. Classifier(FisherClassifier and LeastSquareClassifier) on Data points in Table 2
Enter your choice:
1
Output:- Figure 1 as shown above

[zytham@s158519-vm SMAI]$ python Q3M.py
1. Classifier(FisherClassifier and LeastSquareClassifier) on Data points in Table 1
2. Classifier(FisherClassifier and LeastSquareClassifier) on Data points in Table 2
Enter your choice:
2

Output: - Figure 2 as shown above

5. Difference in the classifier learnt in the above two cases. Give reasons.
   Case 1: - Analysis on data set – Table 1
   Classifier obtained by Least square and Fisher approaches classify data set cleanly. And we obtain a successful classifier. As shown in figure 1 above.

   Case 2: - Analysis on data set – Table 2
   Classifier obtained by Least square does not classify data set cleanly. However, we obtain a successful classifier by Fisher approach. As shown in figure 2 above.

Problem 4: Relation between Least Squares and Fisher' linear Discriminant

Problem 4:

Given m_1 be the No. of points in class C_1 and m_2 in class C_2

label $y_i = \begin{cases} m/m_1 & x_i \in C_1 \\ -m/m_2 & x_i \in C_2 \end{cases}$

RTP:- linear classifier learnet using least square is same as fisher's linear discriminant

i.e: weight vector found co-incides with fisher's criterian.

sum of squares error function can be written as

$$E = \frac{1}{2} \sum_{n=1}^{N} (w^T x_n + w_0 - t_n)^2$$

take derivate w.r.t w_0 & w, first take derivative w.r.t w_0.

$$\frac{\partial E}{\partial w_0} = 0$$

$$\Rightarrow \cancel{\frac{2}{2}} \sum_{n=1}^{N} (w^T x_n + w_0 - t_n) \cdot 1 = 0 \quad \text{(1)}$$

$$\Rightarrow \sum_{n=1}^{N} (w^T x_n + w_0 - t_n) = 0$$

As we know,

$$\sum_{m=1}^{M} t_m = m_1 \cdot \frac{M}{m_1} + m_2\left(-\frac{M}{m_2}\right)$$

$$= \frac{m_1 M}{m_1} - \frac{m_2 M}{m_2}$$

$$= 0$$

$$\therefore \sum_{m=1}^{M} t_m = 0 \qquad \text{--- (III)}$$

μ = mean of given sample $= \frac{1}{M}\sum_{m=1}^{M} x_m = \frac{1}{M}(m_1\mu_1 + m_2\mu_2)$ --- (4)

μ_1 = mean of class C1 samples

μ_2 = mean of class C2 samples.

Now equation (1) can be written as,

$$\sum_{m=1}^{M} (w^T x_m + w_0 - t_m) = 0$$

$$\sum_{m=1}^{M} w^T x_m + \sum_{m=1}^{M} w_0 - \sum_{m=1}^{M} t_m = 0$$

reduces to zero using eqn (3)

$$\therefore -\sum_{m=1}^{M} w^T x_m = \sum_{m=1}^{M} w_0$$

$$w_0 M = -\sum_{m=1}^{M} w^T x_m$$

$$w_0 = -\frac{w^T}{M} \sum_{m=}^{M} x_m$$

Using eqn (4)

$$\boxed{w_0 = -w^T \mu} \quad - (5)$$

Now take partial derivative w.r.t w,

$$\frac{\partial E}{\partial w} = 0 \quad \sum_{m=1}^{M} (w^T x_m + w_0 - t_m) x_m = 0 \quad - (6)$$

As we know from fisher's discriminant

S_w = Within class covariance matrix

$$= \sum_{m \in c_1} (x_m - \mu_1)(x_m - \mu_1)^T + \sum_{m \in c_2} (x_m - \mu_2)(x_m - \mu_2)^T \quad - (7)$$

and

S_B = betweenclass ~~disen~~ covariance Matrix

$$= (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \quad - (8)$$

Considering eqn (6) and write it in terms of S_W & S_B.

$$\sum_{m=1}^{M} w^T x_m x_m + \sum_{m=1}^{M} w_0 x_m - \sum_{m=1}^{M} t_m x_m = 0$$

using $w_0 = -w^T \mu$ and $\sum t_m = M_1$ (class 1) $+ M_2$ (class 2)

$$\sum_{m=1}^{M} w^T x_m x_m + \sum_{m=1}^{M} w_0 x_m = (\mu_1 - \mu_2) M$$

Now write in form of S_B & S_W

$$\left[\frac{S_W}{M} + \frac{m_1 m_2}{M^2}(\mu_1 - \mu_2)(\mu_1 - \mu_2)^t\right] w = (\mu_1 - \mu_2)$$

→ (9)

since, vector $(\mu_1 - \mu_2)(\mu_1 - \mu_2)^t w$ is in direction of $(\mu_1 - \mu_2)$ for any value of w

$$\therefore \frac{m_1 m_2}{M^2}(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T w = (1 - \alpha)(\mu_1 - \mu_2)$$

Using above in eqn (9)

$$w = \alpha M S_W^{-1}(\mu_1 - \mu_2)$$

It is identical to solution for Fisher's linear discriminant.

i.e: Weight vector found co-incides with fisher's discriminant.