ASSIGNMENT 1

Dipankar Niranjan
201503003

CODE:

```
from random import randrange
import matplotlib.pyplot as plt
from copy import deepcopy
from math import sqrt

class1 = [[1, 2, 7], [1, 8, 1], [1, 7, 5], [1, 6, 3], [1, 7, 8], [1, 5, 9], [1, 4, 5]]
class2 = [[1, 4, 2], [1, -1, -1], [1, 1, 3], [1, 3, -2], [1, 5, 3.25], [1, 2, 4], [1, 7, 1]]

global line1
global line2
global line3
global line4

def singlesampleperceptron():
    # generate w matrix
    weight = []
    for i in range(0,3):
            weight.append(randrange(-10,11))

    # normalize class2
    for i in class2:
            for j in range(len(i)):
                    i[j] = -i[j]

    # create dataset
    dataset = []
    for i in class1:
            dataset.append(i)
    for i in class2:
            dataset.append(i)

    # main loop
    k = 0
    count = 0
    while 1:
            value = 0
            for i in range(0,3):
                    value += weight[i] * dataset[k][i]

            if value > 0:
                    count += 1
                    if count == len(dataset):
                            break
            else:
                    count = 0
                    for i in range(0,3):
                            weight[i] += dataset[k][i]

            k += 1
            if k == len(dataset):
                    k %= len(dataset)

    print weight
```

```
    # denormalize
    for i in class2:
            for j in range(len(i)):
                    i[j] = -i[j]

    class12Dx = []
    class12Dy = []
    class22Dx = []
    class22Dy = []
    for i in range(len(class1)):
            class12Dx.append(class1[i][1])
            class12Dy.append(class1[i][2])
    for i in range(len(class2)):
            class22Dx.append(class2[i][1])
            class22Dy.append(class2[i][2])

    plt.plot(class12Dx, class12Dy, 'bo')
    plt.plot(class22Dx, class22Dy, 'ro')

    # find points on x and y axes
    y1 = -(weight[0]/weight[2])
    x2 = -(weight[0]/weight[1])

    line1, = plt.plot([0, y1], [x2, 0], label = "ssp")
    plt.setp(line1, color='r', linewidth=1.0)

    return line1

def singlesampleperceptronmargin(margin):
    # generate w matrix
    weight = []
    for i in range(0,3):
            weight.append(randrange(-10,11))

    # normalize class2
    for i in class2:
            for j in range(len(i)):
                    i[j] = -i[j]

    # create dataset
    dataset = []
    for i in class1:
            dataset.append(i)
    for i in class2:
            dataset.append(i)

    # main loop
    k = 0
    count = 0
    while 1:
            value = 0
            for i in range(0,3):
                    value += weight[i] * dataset[k][i]

            if value > margin:
                    count += 1
                    if count == len(dataset):
                            break
            else:
                    count = 0
                    for i in range(0,3):
```

```
                                        weight[i] += dataset[k][i]

                k += 1
                if k == len(dataset):
                        k %= len(dataset)

        print weight

        # denormalize
        for i in class2:
                for j in range(len(i)):
                        i[j] = -i[j]

        class12Dx = []
        class12Dy = []
        class22Dx = []
        class22Dy = []
        for i in range(len(class1)):
                class12Dx.append(class1[i][1])
                class12Dy.append(class1[i][2])
        for i in range(len(class2)):
                class22Dx.append(class2[i][1])
                class22Dy.append(class2[i][2])

        plt.plot(class12Dx, class12Dy, 'bo')
        plt.plot(class22Dx, class22Dy, 'ro')

        # find points on x and y axes
        y1 = -(weight[0]/weight[2])
        x2 = -(weight[0]/weight[1])

        line2, = plt.plot([0, y1], [x2, 0], label='sspm')
        plt.setp(line2, color='b', linewidth=1.0)

        return line2

def relaxationalgo(lrate, margin):
        weight = [0,0,0]
        weight[0] = 1
        weight[1] = 1
        weight[2] = 1

        # normalize class2
        for i in class2:
                for j in range(len(i)):
                        i[j] = -i[j]

        # create dataset
        dataset = []
        for i in class1:
                dataset.append(i)
        for i in class2:
                dataset.append(i)

        # main loop
        k = 0
        count = 0
        overallcount = 0
        while 1:
                overallcount += 1
                value = 0
                for i in range(0,3):
```

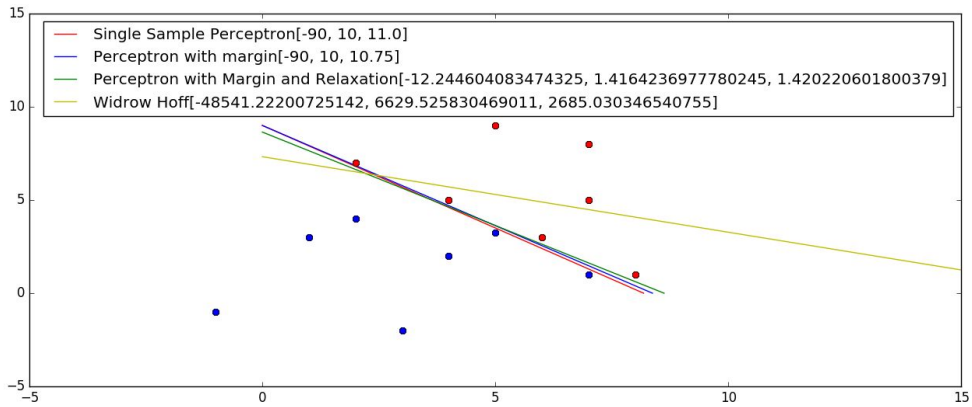
```
                        value += weight[i] * dataset[k][i]
                if value > margin:
                        count += 1
                        if count == len(dataset):
                                break
                else:
                        count = 0

                        value = 0
                        for i in range(0,3):
                                value += (weight[i] * dataset[k][i])
                        value = (-(value) + margin)/((dataset[k][0]*dataset[k][0]) + (dataset[k][1]*dataset[k][1]) +
(dataset[k][2]*dataset[k][2]))

                        for j in range(0,3):
                                weight[j] = weight[j] + (lrate * value * dataset[k][j])

                k += 1
                if k == len(dataset):
                        k %= len(dataset)

    print weight

    # denormalize
    for i in class2:
                for j in range(len(i)):
                        i[j] = -i[j]

    class12Dx = []
    class12Dy = []
    class22Dx = []
    class22Dy = []
    for i in range(len(class1)):
                class12Dx.append(class1[i][1])
                class12Dy.append(class1[i][2])
    for i in range(len(class2)):
                class22Dx.append(class2[i][1])
                class22Dy.append(class2[i][2])

    plt.plot(class12Dx, class12Dy, 'bo')
    plt.plot(class22Dx, class22Dy, 'ro')

    # find points on x and y axes
    y1 = -(weight[0]/weight[2])
    x2 = -(weight[0]/weight[1])

    line3, = plt.plot([0, y1], [x2, 0], label='relaxationalgo')
    plt.setp(line3, color='g', linewidth=1.0)

    return line3

def widrowhoff(lrate, theta):
    bvec = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]

    weight = [0,0,0]
    weight[0] = 1
    weight[1] = 1
    weight[2] = 1

    # normalize class2
    for i in class2:
                for j in range(len(i)):
```

```
                        i[j] = -i[j]

# create dataset
dataset = []
for i in class1:
        dataset.append(i)
for i in class2:
        dataset.append(i)

# main loop
k = 0
count = 1
overallcnt = 0
while 1:
        value = 0
        for i in range(0,3):
                value += weight[i] * dataset[k][i]
        value = bvec[k] - value

        lr = lrate/count

        temp = deepcopy(dataset[k])
        for i in range(len(temp)):
                temp[i] = temp[i] * value * lr

        if sqrt((temp[0]*temp[0]) + (temp[1]*temp[1]) + (temp[2]*temp[2])) < theta:
                overallcnt += 1
                if overallcnt == 1:
                        break
        else:
                overallcnt = 0
                for j in range(0,3):
                        weight[j] = weight[j] + temp[j]

        k += 1
        if k == len(dataset):
                k %= len(dataset)
        count += 1

print weight

# denormalize
for i in class2:
        for j in range(len(i)):
                i[j] = -i[j]

class12Dx = []
class12Dy = []
class22Dx = []
class22Dy = []
for i in range(len(class1)):
        class12Dx.append(class1[i][1])
        class12Dy.append(class1[i][2])
for i in range(len(class2)):
        class22Dx.append(class2[i][1])
        class22Dy.append(class2[i][2])

plt.plot(class12Dx, class12Dy, 'bo')
plt.plot(class22Dx, class22Dy, 'ro')

# find points on x and y axes
y1 = -(weight[0]/weight[2])
```

```
    x2 = -(weight[0]/weight[1])

    line4, = plt.plot([0, y1], [x2, 0], label='widrowhoff')
    plt.setp(line4, color='y', linewidth=1.0)

    return line4

def main():
    line1 = singlesampleperceptron()

    margin = 0.5
    line2 = singlesampleperceptronmargin(margin)

    lrate = 2
    margin = 0.5
    line3 = relaxationalgo(lrate, margin)

    lrate = 0.7
    theta = 0.01
    line4 = widrowhoff(lrate, theta)

    plt.axis([-10, 10, -10, 10])
    plt.legend([line1, line2, line3, line4], ['Single Sample Perceptron', 'Perceptron with margin', 'Perceptron with Margin and
Relaxation', 'Widrow Hoff'])
    plt.show()

if __name__ == '__main__':
    main()
```

1 l)

GRAPH:



a (init) = [1,1,1]
SSP
a = [-90, 10, 11.0]
SSPM with a margin = 0.5
a = [-90, 10, 10.75]
PMR with eta = 2 and margin = 0.5
a = [-12.244604083474325, 1.4164236977780245, 1.420220601800379]
Widrow Hoff eta = 0.7 and theta = 0.01

a = [-48541.22200725142, 6629.525830469011, 2685.030346540755]

1 II)
Weight Initialization vs Time taken (in sec)

w[0,1,2]	**[-90, 10,11]**	**[-10,1,1]**	**[0,0,0]**	**[1,1,1]**	**[10,10,10]**	**[-10,-10,-10]**
SSP	3.504e-05	0.000641	0.000735	0.002391	0.000840	0.000202
SSPM	3.409e-05	0.002244	0.001023	0.002406	0.001377	0.001358
RAM	2.694e-05	0.002343	0.007210	0.007333	0.000132	0.000236
WH	1.295633	0.446901	0.242027	1.634860	13.04498	13.09362

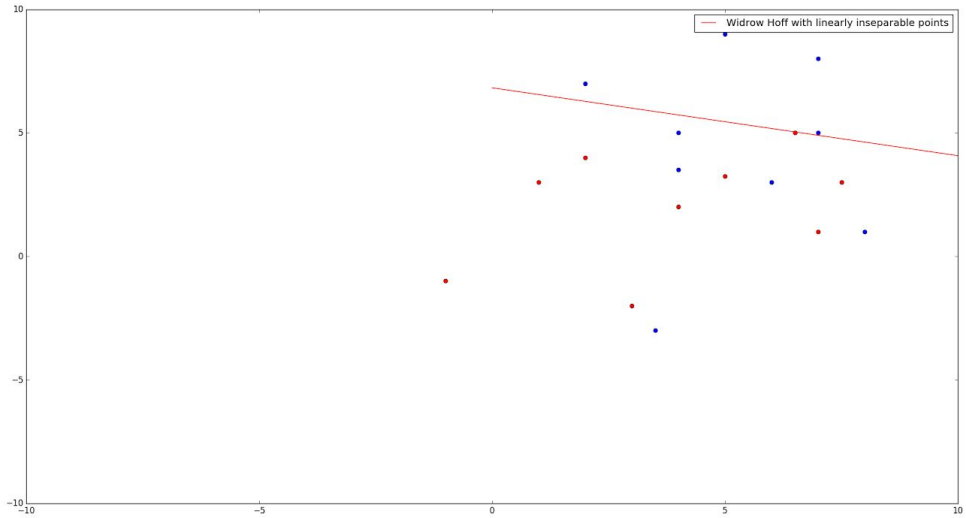
We can see that the convergence is quicker if the initialized weight vectors have value close to what the final weight values are (depending on - from which direction the convergence occurs)

1 III)

b	**0.5**	**0.25**	**0.1**	**0.75**	**1**
SSPM	**0.002406** [-90, 10, 10.75]	**0.002918** [-100,12,11.75]	**0.002837** [-100,12,11.75]	**0.002748** [-100,11,13.5]	**0.001050** [-58,7,6.75]
RAM	**0.007333** [-12.244, 1.416, 1.420]	**0.006256** [-6.121, 0.710, 0.710]	**0.002463** [-2.448,0.283,0.284]	**0.007403** [-18.365,2.129,2.132]	**0.007176** [-24.487,2.847,2.844]

The ‘a’ vector was initialized as [1,1,1] in all the cases.
As b increases, the number of iterations required to get the initialized weight vector to satisfy the condition increases, as a result time increases, unless of course increasing b results in a convergence at an earlier point in space - which in turn means less time taken.

1 IV)



We can see that the Widrow Hoff method misclassifies only five of the blue points out of a total of 18 points which is a reasonably acceptable error, for the given value of theta = 0.01 and given locations of the points in 2D space.

1 V) Code for all the questions has been provided above

Single Sample Perceptron - Used to classify linearly separable data. Checks if all the datapoints (class 2 normalized) satisfy the equation $a^t y_i >= 0$ for all points i. We iterate over all the points and update the weight vector until the above equation is satisfied by all the points. We can see that all the points are correctly classified into their respective classes.

Single Sample Perceptron with margin - This is basically the same as the above algorithm except all points now have to satisfy $a^t y_i >= b >= 0$ where b is a positive constant. Same procedure as the above algorithm and a result where all points are at a certain distance (specified by b) from the class boundary.

Relaxation algorithm with margin - This algorithm is again single sample, but we update the weight vector only when we encounter a misclassified point. The update is proportional to the error in classification, given by the error function J. We move in the direction opposite to the gradient (derivative of J with respect to ‘a’) to minimize the error function. We update the weights proportional to a learning rate eta. The error function gives the relaxation factor.

Widrow Hoff procedure - This algorithm looks similar to Relaxation algorithm but it stops when the norm calculated is lesser than a constant theta for any point. We don’t need to do this for all points. Again we update the weight vector only when there is a misclassification occurring. The relaxation rule is a correction rule here and this procedure works very well in yielding an approximate linear classifier when the dataset is not linearly classifiable. Also we anneal the learning rate eta here for convergence to occur.