STATISTICAL METHODS IN AI
ASSIGNMENT2:
Linear Discriminant Functions

Name – Megha Agarwal
Roll No – 201506511
Course – M.Tech CSIS(PG1)
Dated : 1 February, 2016

AIM:

The idea of this assignment is to explore the material presented as part of Chapter 5 on constructing Linear Discriminant Functions based on various approaches such as Perceptron criterion function and Mean Squared Error.

ALGORITHM (GENERAL CONCEPT):

Generalised Linear Discriminant Functions:

The linear discriminant function g(x) can be written as

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i,$$

By continuing to add terms we can obtain the class of polynomial discriminant functions. These can be thought of as truncated series expansions of some arbitrary g(x), and this in turn suggest the generalized linear discriminant function.

$$g(\mathbf{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(\mathbf{x})$$

$$g(x) = a^t y$$

The Two-Category Linearly-Separable Case

Suppose now that we have a set of n samples y 1 , ..., y n , some labelled ω 1 and some labelled ω 2 . We want to use these samples to determine the weights a in a linear discriminant function g(x) = a t y. We tend to look for a weight vector that classifies all of the samples correctly. If such a weight vector exists, the samples are said to be linearly separable.

Generalise Classification:

If $\underline{a}^t \underline{y} > 0$: It is labelled as class ω 1

If $\underline{a}^t \underline{y} < 0$: then It is labelled as class ω 2

If $\underline{a}^t \underline{y} = 0$: The points are on the solution vector

Normalization:

It simplifies the treatment of the two-category case, viz., the replacement of all samples labelled ω 2 by their negatives.
As a result, our classification will be as :

If $\underline{a}^t \underline{y} > 0$: It is properly classfied .

If $\underline{a}^t \underline{y} < 0$: It is missclassified

Assumption :

If $\underline{a}^t \underline{y} = 0$ i.e. ***For the point on the solution vector, the algo implemented considers it to be classified.***

Output:

The solution is weight vector which is called a separating vector or more generally a solution vector. It divides the n samples into their respectives class (ω 1 and ω 2).

Theory Question:

1. Prove that the single-sample perceptron algorithm will always converge to a solution, if one exists (List out all the assumptions, make each step explicit and mathematically precise, but not be verbose! Do not copy directly from the textbook).

Perceptron Convergence Theorem :

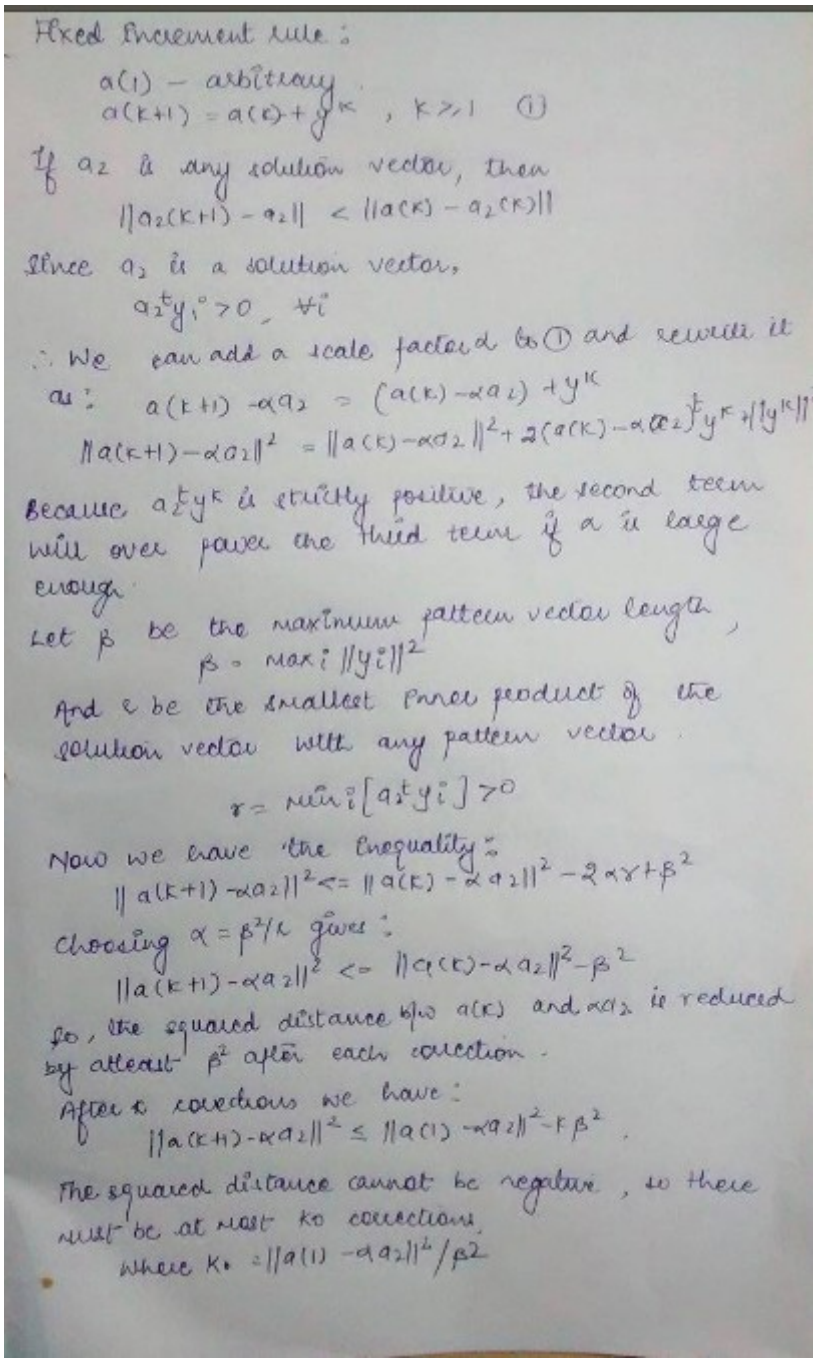
Solving a two class problem:
Let us assume there are two class c1 and c2. These classes are assumed to be linearly separable.
The classes are existing in multidimensional space.
The hyperplane will separate the two classes.

To Prove : Single Sample perceptron algorithm will always converge to a solution if the training sample are linearly separable.

Assumptions:

1. Linear Separability.
2. w(0), the weight vector is initialised by 0.
3. n (k) is constant – there is fixed increment.
4. We keep on feeding the patterns X(n).
   n = 1, 2, ...
5. Initially we assume when we feed x1, x2 .....that there is a misclassification. When there is an error, there is learning.

6. Whenever a single sample is misclassified we modify the weight vector.
6. If some value of exists that gives correct classification, then we can say that system is convergent.



Fixed Increment rule :

$a(1)$ – arbitrary

$a(k+1) = a(k) + y^k$, $k \geq 1$ ①

If a_2 is any solution vector, then

$\|a_2(k+1) - a_2\| < \|a(k) - a_2(k)\|$

Since a_2 is a solution vector,

$a_2^t y_i > 0$, $\forall i$

∴ We can add a scale factor to ① and rewrite it as :

$a(k+1) - \alpha a_2 = (a(k) - \alpha a_2) + y^k$

$\|a(k+1) - \alpha a_2\|^2 = \|a(k) - \alpha a_2\|^2 + 2(a(k) - \alpha a_2)^t y^k + \|y^k\|^2$

Because $a_2^t y^k$ is strictly positive, the second term will over power the third term if α is large enough.

Let β be the maximum pattern vector length,

$\beta = \max_i \|y_i\|^2$

And γ be the smallest inner product of the solution vector with any pattern vector.

$\gamma = \min_i [a_2^t y_i] > 0$

Now we have the inequality :

$\|a(k+1) - \alpha a_2\|^2 <= \|a(k) - \alpha a_2\|^2 - 2\alpha\gamma + \beta^2$

Choosing $\alpha = \beta^2/\gamma$ gives :

$\|a(k+1) - \alpha a_2\|^2 <= \|a(k) - \alpha a_2\|^2 - \beta^2$

So, the squared distance b/w $a(k)$ and αa_2 is reduced by atleast β^2 after each correction.

After k corrections we have :

$\|a(k+1) - \alpha a_2\|^2 \leq \|a(1) - \alpha a_2\|^2 + k\beta^2$.

The squared distance cannot be negative, so there must be at most k_0 corrections.

where $k_0 = \|a(1) - \alpha a_2\|^2 / \beta^2$

The weight vector that results after the corrections are done must classify all samples correctly, because there are a finite number of corrections that only occur at each missclassification.

Hence k0 is a bound on the number of corrections, and there will always be a finite number of corrections using the fixed increment rule if the samples are linearly separable.

PRACTICAL EXCERCISES:

The data set to be used for the exercises given below is the following sample set comprising a two-class problem.
wClass 1 = [(1; 6); (7; 2); (8; 9); (9; 9); (4; 8); (8; 5)]
wClass 2 = [(2; 1); (3; 3); (2; 4); (7; 1); (1; 3); (5; 2)]

CODE (FOR QUESTIONS 2, 3, 4, 5, 6 & 7)
(Choices to be Selected for the answer of Selected Question)

```
import matplotlib.pyplot as plt
import numpy as np
import pylab as pl
import math as math

def augment_vector(X):
        bias = np.ones((len(X), 1))
        return np.hstack((X, bias))

def normalise_train_set(X, Y):
        X = np.asarray(X)
        Y = np.asarray(Y)
        train_set = augment_vector(X)
        labels_ =  np.unique(Y)
        idx = Y==labels_[1]                                  #Select one class as negative class
        train_set[idx] = -train_set[idx]#Make the x coordinate of selected class as negative
        dim = train_set.shape[1]   #Return Dimensionality of feature space // train set
        return X, Y, train_set, dim

def single_sample_perceptron(X, Y, learning_rate, margin):
        #Obtained the values as required (Augmented and negated)
        X, Y, train_set, dim = normalise_train_set(X, Y)

        i=0
        n = len(train_set)
        k=0
        weights = [0, 0, 1]
        print weights
        count = 0
        while i!=n:
                k = (k+1)%n
                count+=1
                if np.dot(train_set[k],weights)<=margin:
                        weights = weights + learning_rate * train_set[k]
                        i=0
                else:
                        i=i+1
        print 'Converged after {} iterations'.format(count)
        print weights
        plot_boundary(weights, train_set, X, Y)
```

```
    return weights

def relaxation_algo_with_margin(X, Y, learning_rate, margin):
    #Obtained the values as required (Augmented and negated)
    X, Y, train_set, dim = normalise_train_set(X, Y)
    weights = [0, 0, 1]
    k=-1
    i=0
    count=0
    while i!=len(train_set):
        k=(k+1)%len(train_set)
        if np.dot(train_set[k],weights)<=margin:
            i=0
            temp1=(margin - np.dot(train_set[k],weights))
            temp2=np.dot(train_set[k],train_set[k])
            temp3=float((float(temp1)/float(temp2))*2)
            temp=np.dot(temp3,train_set[k])
            weights=weights + (learning_rate * temp)
        else:
            i+=1
            count+=1

    print weights
    print 'Converged after {} iterations'.format(count)
    plot_boundary(weights, train_set, X, Y)
    return weights

def least_mean_square(X, Y, learning_rate, margin):
    #Obtained the values as required (Augmented and negated)
    X, Y, train_set, dim = normalise_train_set(X, Y)

    a4=[0.5,0.5,-1]
    a4=[0.125,0.125,-1]
    k=-1
    n=len(train_set)
    cnt=0
    nk1=1
    count=0
    while 1:
        k=(k+1)%n
        flag=1
        count+=1
        nk=float(nk1)/2000.0
        temp1=nk* (margin - np.dot(train_set[k],a4))
        temp2=np.dot(temp1,train_set[k])
        temp3=math.sqrt(np.dot(temp2,temp2))
        if temp3<margin:
            flag=0
        a4=np.sum([a4,temp2],axis=0)
        if(flag==1 or count==1000):
```

```
                break;

        print a4
        plot_boundary(a4, train_set, X, Y)
        return a4

def ques_6_least_mean_square(X, Y, learning_rate, margin):
        #Obtained the values as required (Augmented and negated)
        X, Y, train_set, dim = normalise_train_set(X, Y)

        a4=[0.5, 0.5 ,-1]
        k=-1
        n=len(train_set)
        cnt=0
        nk1=1
        count=0
        while 1:
                k=(k+1)%n
                flag=1
                count+=1
                nk=float(nk1)/2000.0
                temp1=nk* (margin - np.dot(train_set[k],a4))
                temp2=np.dot(temp1,train_set[k])
                temp3=math.sqrt(np.dot(temp2,temp2))
                if temp3<margin:
                        flag=0
                a4=np.sum([a4,temp2],axis=0)
                if(flag==1 or count==2000):
                        break;

        print a4
        weights = single_sample_perceptron_ques_6(X, Y, learning_rate, margin)
        plot_boundary_ques_6(a4, weights, train_set, X, Y)
        return a4

def plot_boundary_ques_6(weights1, weights2, train_set, X, Y):
        # plot data-points
        x_points = X[:,0]
        y_points = X[:,1]
        length = len(x_points)

        x_points_1 = x_points[0:length/2]
        x_points_2 = x_points[length/2:length]
        y_points_1 = y_points[0:length/2]
        y_points_2 = y_points[length/2:length]

        plt.plot(x_points_1,y_points_1,'ro');
        plt.axis([0,10,0,10])
        plt.plot(x_points_2,y_points_2,'bo');
```

```
	a,b,c = weights1
	xchord_1 = 0
	xchord_2 = -(float(c))/(float(a))
	ychord_2 = 0
	ychord_1 = -(float(c))/(float(b))

	Lms_Line, =plt.plot([xchord_1, xchord_2],[ychord_1, ychord_2],'black', label='Lms Line')
	a,b,c = weights2
	xchord_1 = 0
	xchord_2 = -(float(c))/(float(a))
	ychord_2 = 0
	ychord_1 = -(float(c))/(float(b))

	Perceptron_Line, = plt.plot([xchord_1, xchord_2],[ychord_1, ychord_2], 'Green', label='Perceptron Line')
	plt.legend([Lms_Line, Perceptron_Line],['Lms Line', 'Perceptron Line'])
	plt.show()
def single_sample_perceptron_ques_6(X, Y, learning_rate, margin):
	#Obtained the values as required (Augmented and negated)
	X, Y, train_set, dim = normalise_train_set(X, Y)

	i=0
	n = len(train_set)
	k=0
	weights = [1, 1, 1]
	print weights
	count = 0
	while i!=n:
		k = (k+1)%n
		count+=1
		if np.dot(train_set[k],weights)<=margin:
			weights = weights + learning_rate * train_set[k]
			i=0
		else:
			i=i+1
	print weights
	return weights

def plot_boundary(weights, train_set, X, Y):
	# plot data-points
	x_points = X[:,0]
	y_points = X[:,1]
	length = len(x_points)

	x_points_1 = x_points[0:length/2]
	x_points_2 = x_points[length/2:length]
	y_points_1 = y_points[0:length/2]
	y_points_2 = y_points[length/2:length]
```

```
        plt.plot(x_points_1,y_points_1,'ro');
        plt.axis([0,10,0,10])
        plt.plot(x_points_2,y_points_2,'bo');


        a,b,c = weights
        xchord_1 = 0
        xchord_2 = -(float(c))/(float(a))
        ychord_2 = 0
        ychord_1 = -(float(c))/(float(b))

        plt.plot([xchord_1, xchord_2],[ychord_1, ychord_2],'black')
        plt.show()

def predict(test_set, weights):
        test_set = augment_vector(test_set)
        print weights
        pred_list = []
        for i in range(len(test_set)):
                print test_set[i]
                if np.dot(test_set[i], weights)<0:
                        pred_list.append(2)
                elif np.dot(test_set[i], weights)>0:
                        pred_list.append(1)

        return pred_list



def compute_accuracy(pred_labels_, Y_test):
        count=0
        length = len(pred_labels_)
        for k in range(len(pred_labels_)):
                if pred_labels_[k]==Y_test[k]:
                        count = count+1

        accuracy = (float(count)/float(length))*100
        return accuracy


if __name__ == '__main__':
        X= [(1, 6), (7, 2), (8, 9), (9, 9), (4, 8), (8, 5), (2, 1), (3, 3), (2, 4), (7, 1), (1, 3), (5,
2)]
        Y  = [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2]
        testset = [(0, 1), (8, 1), (2, 6), (2, 4.5), (6, 1.5), (4, 3)]
        test = [(6 , 4) ,(6, 6), (9 , 4),  (0, 0),  (0 ,-2) , (1, 1) ]
        Y_test = [ 1, 1, 1, 2, 2, 2]
        Xnew= [(1, 6), (7, 6), (8, 9), (9, 9), (4, 8), (8, 5), (2, 1), (3, 3), (2, 4), (7, 1), (1,
3), (5, 2)]
```

```
X_no_sep= [(2, 1), (7, 2), (2, 4), (9, 9), (4, 8), (5, 2),(1, 6), (3, 3), (8, 9), (7, 1), (1, 3), (8, 5)]

print 'Enter the type of algo to follow: '
print "Enter 1.....Single-sample perceptron"
print "Enter 2.....Single-sample perceptron with margin"
print "Enter 3.....Relaxation algorithm with margin"
print "Enter 4.....Widrow-Hoff or Least Mean Squared (LMS) Rule"
print "Enter 5.....Solutions Algin Case"

choice = input("Enter Choice\n")

if choice == 1:
        learning_rate = 1.0
        margin = 0.0
        weights = single_sample_perceptron(X, Y, learning_rate, margin)
        pred_labels = predict(test, weights)
        print "Predicted Classes: ",
        print pred_labels
        accuracy = compute_accuracy(pred_labels, Y_test)
        print 'Accuracy for the test data set is  {}%'.format(accuracy)


elif choice == 2:
        learning_rate = 1.0
        margin = 2.0
        weights = single_sample_perceptron(X, Y, learning_rate, margin)
        pred_labels = predict(test, weights)
        print "Predicted Classes: ",
        print pred_labels
        accuracy = compute_accuracy(pred_labels, Y_test)
        print 'Accuracy for the test data set is  {}%'.format(accuracy)


elif choice == 3:
        learning_rate = 1.0
        margin = 1.0
        weights = relaxation_algo_with_margin(X, Y, learning_rate, margin)
        pred_labels = predict(test, weights)
        print "Predicted Classes: ",
        print pred_labels
        accuracy = compute_accuracy(pred_labels, Y_test)
        print 'Accuracy for the test data set is  {}%'.format(accuracy)

        print 'Non Separable Data Plotting is not possible!!'
        #weights = relaxation_algo_with_margin(X_no_sep, Y, learning_rate, margin)


elif choice==4:
```

```
            learning_rate =0.01
            margin = 1.0
            print 'Original Plot'
            weights = least_mean_square(X, Y, learning_rate, margin)
            print 'Test Data Set'
            pred_labels = predict(test, weights)
            print "Predicted Classes: ",
            print pred_labels
            accuracy = compute_accuracy(pred_labels, Y_test)
            print 'Accuracy for the test data set is  {}%'.format(accuracy)
            print 'Non Separable Data Plotting'
            weights = least_mean_square(X_no_sep, Y, learning_rate, margin)

    elif choice==5:
            learning_rate =0.01
            margin = 1.0
            print 'Properly Classified : Solutions Align : Q-6'
            X= [(1, 6), (7, 2), (8, 9), (9, 9), (4, 8), (8, 5), (2, 1), (3, 3), (2, 4), (7, 1), (1,
3), (5, 2)]
            Xnew= [(1, 5.6), (7, 4.3), (8, 9), (9, 9), (4, 8), (8, 4.4), (2, 1), (3, 3), (2, 4),
(7, 0.5), (1, 3), (5, 2)]
            weights = ques_6_least_mean_square(Xnew, Y, learning_rate, margin)
```

Q-2 SINGLE SAMPLE PERCEPTRON

Algorithm:

```
begin
init a, k= 0
do
    k = (k+ 1) mod n
    if yk is misclassified by a
        then a = a + yk
    until all patterns classified
return a
end
```

The single sample perceptron algorithm takes any random misclassified sample and compute the value of J(a) – the criterion functions. The algorithm tends to minimise the value of the criterion function and returns the value of a for which the value of criterion function is minimum.

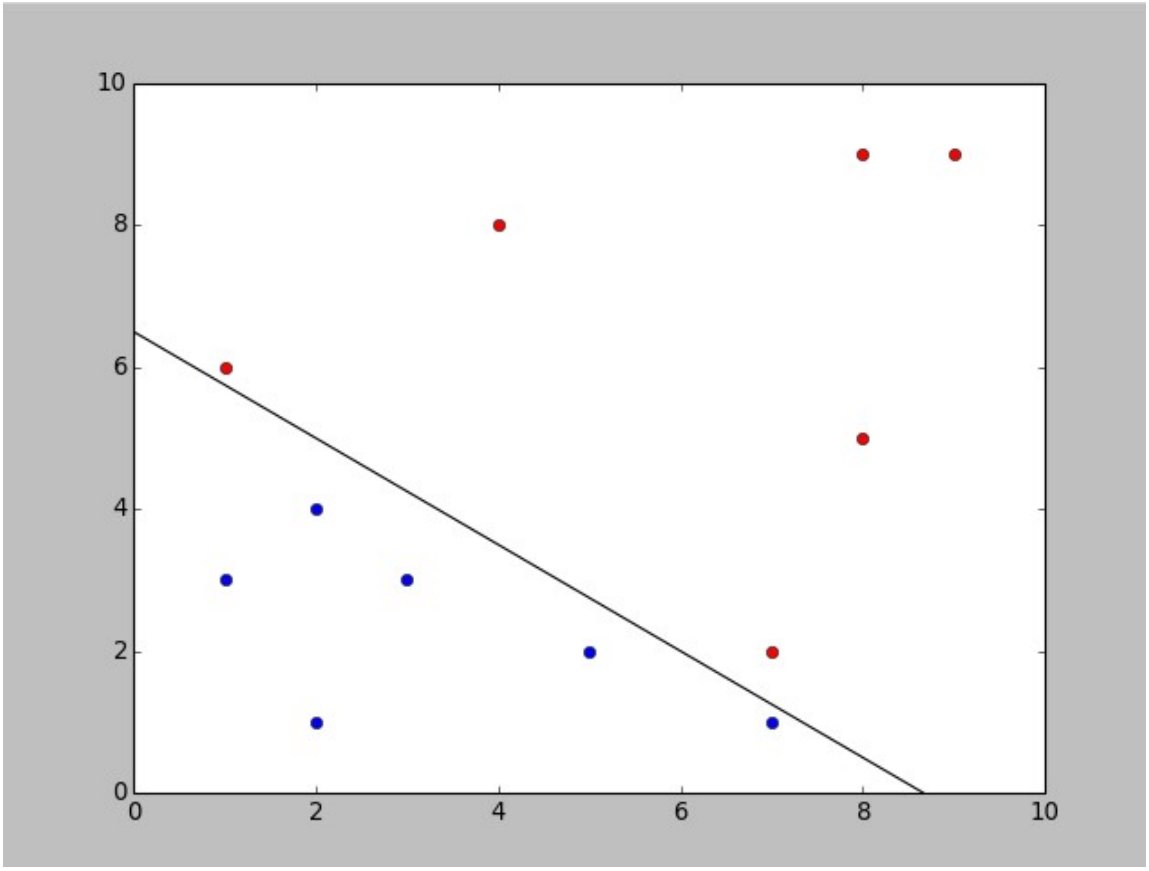
2. A) Plot the data points in a graph (e.g. Circle: class-1 and Cross: class-2) and also show the weight vector a learnt from all of the above algorithms in the same graph (labeling clearly to distinguish different solutions).

Assumptions:
Plotted Class 1 with Red Circles.
Plotted Class2 with Blue Circles.

The points on the solution vector is considered to be classified. Hence Accuracy here is 100%.
Learning Rate – 1.0
Margin - 0.0 (Without margin).
Initial weight vector – [0, 0, 1] (Can be taken as random as well).
Final weight vector - [3. 4. -26.] (Plotted as Black Line)

Sample Output:

```
Enter the type of algo to follow:
Enter 1.....Single-sample perceptron
Enter 2.....Single-sample perceptron with margin
Enter 3.....Relaxation algorithm with margin
Enter 4.....Widrow-Hoff or Least Mean Squared (LMS) Rule
Enter Choice : 1
Converged after 391 iterations
[ 3.  4. -26.]
```



2 B) Create a test set comprising three more data samples (y i) for each class and test your implementation by computing for the test samples the output (class label) predicted by the algorithm.

Test Set:

Test Set : [(0, 1), (8, 1), (2, 6), (2, 4.5), (6, 1.5), (4, 3)]

Corresponding Class Labels :
[1,1,1,2,2,2]

Output:
[2 1 1 2 2 2]
Accuracy for the test data set is 83.3333333333%

2 C) Run each of the above algorithms for various initial values of the weight vector, and comment on the dependence of convergence time (run-time) on initialization.

Assumptions:
The convergence time(run time) is measured in terms of the number of iterations the algorithm took to reach to the final solution vector.
The No of iterations are increasing proportionally wrt. the weight vector taken

Analysis on Various Initial Weight Vectors:

Weight Vector(Initial)	Weight Vector(Final)	No. of Iterations
[0, 0, 1]	[3. 4. -26.]	391 Iterations
[1, 1, 1]	[3. 4. -26.]	427 Iterations
[1, 0, 1]	[3. 4. -26.]	415 Iterations
[0, 1, 1]	[3. 4. -26.]	403 Iterations
[0.75060091 0.96482557 0.52564365]	[2.75060091 3.96482557 -25.47435635]	439 Iterations
[0.69950012 0.14534138 0.9634146]	[2.69950012 4.14534138 -25.0365854]	403 Iterations
[50, 50, 50]	[2. 3. -18.]	607 Iterations
[70, 70, 70]	[3. 4. -26.]	871 Iterations
[100, 100,100]	[3. 4. -26.]	991 Iterations

Analysis : *As the weight vector is deviating more from the final solution vector, it is taking more time/no of iterations to converge to the final solution.*

Q- 3. SINGLE-SAMPLE PERCEPTRON WITH MARGIN

Algorithm:

begin
initialize a, criterion θ, margin b, $\eta(\cdot)$, k = 0
do
$k \leftarrow k + 1$
if $a^t y_k + b < 0$
then $a \leftarrow a - \eta(k) y_k$

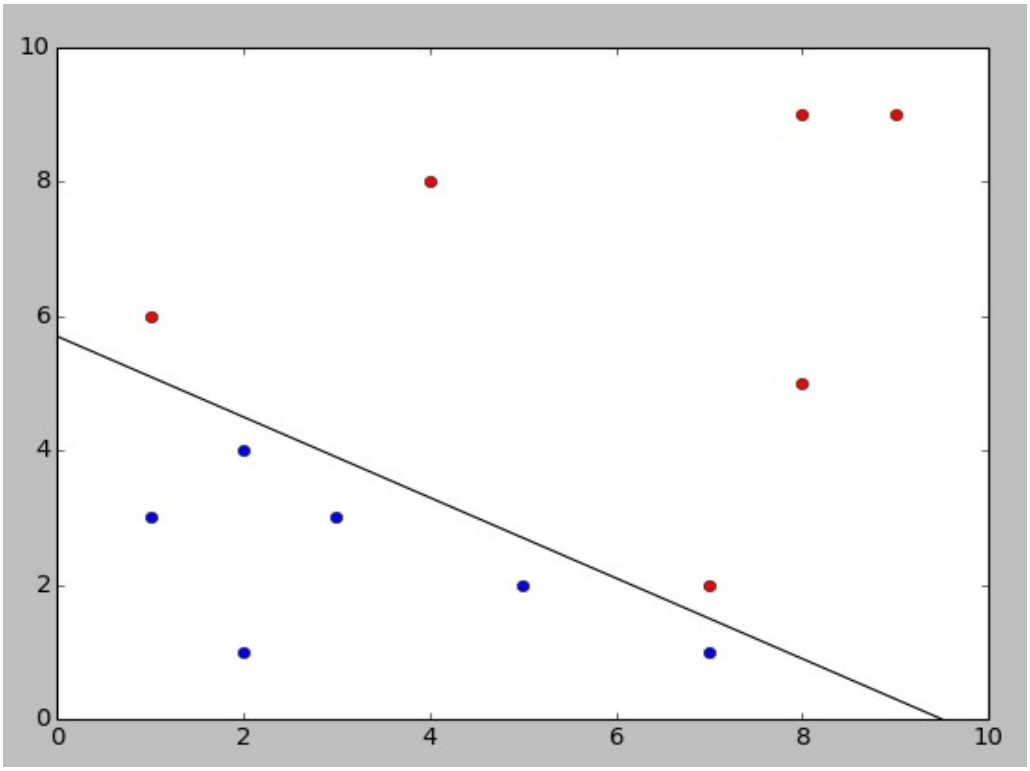
until $a^t y_k + b \leq 0$ for all k
return a
end

The single sample perceptron algorithm takes any random misclassified sample and compute the value of J(a) – the criterion functions. The algorithm tends to minimise the value of the criterion function and returns the value of a for which the value of criterion function is minimum.

3. A) Plot the data points in a graph (e.g. Circle: class-1 and Cross: class-2) and also show the weight vector a learnt from all of the above algorithms in the same graph (labeling clearly to distinguish different solutions).

Assumptions:
Plotted Class 1 with Red Circles.
Plotted Class2 with Blue Circles.
The points on the solution vector is considered to be classified. Hence Accuracy here is 100%.
Learning Rate – 1.0
Margin - 2.0 (Initial Value of margin).
Initial weight vector – [0, 0, 1] (Can be taken as random as well).
Final weight vector - [6. 10. -57.] (Plotted as Black Line)

Sample Output:
Enter the type of algo to follow:
Enter 1.....Single-sample perceptron
Enter 2.....Single-sample perceptron with margin
Enter 3.....Relaxation algorithm with margin
Enter 4.....Widrow-Hoff or Least Mean Squared (LMS) Rule
Enter Choice
2
Converged after 1268 iterations
[6. 10. -57.]



3 B) Create a test set comprising three more data samples (y i) for each class and test your implementation by computing for the test samples the output (class label) predicted by the algorithm.

Test Set:
Test Set : [(6 , 4) ,(6, 6), (9 , 4), (0, 0), (0 ,-2) , (1, 1)]

Corresponding Class Labels :
[1,1,1,2,2,2]

Output:
[1 1 1 2 2 2]
Accuracy for the test data set is 100.0%

3 C) Run each of the above algorithms for various initial values of the weight vector, and comment on the dependence of convergence time (run-time) on initialization.

Assumptions:
The convergence time(run time) is measured in terms of the number of iterations the algorithm took to reach to the final solution vector.
The No of iterations are increasing proportionally wrt. the weight vector taken

Analysis on Various Initial Weight Vectors:

Weight Vector(Initial)	No. of Iterations
[0, 0, 1]	Converged after 1268 iterations
[0, 1, 1]	Converged after 1244 iterations
[1 ,1 ,1]	Converged after 1244 iterations
[5, 5, 5]	Converged after 1268 iterations
[10, 10, 10]	Converged after 1316 iterations

Analysis : *As the weight vector is deviating more from the final solution vector, it is taking more time/no of iterations to converge to the final solution.*

3 D)Similarly explore the effect of adding different margins on the final solution as well as on the convergence (run-) time for algorithms (3) and (4).

Inital Weight Vector: [0, 0, 1]

Margin 1.0:
Converged after 704 iterations
Margin 2.0:
Converged after 1268 iterations
Margin 4.0:
Converged after 1592 iterations
Margin 5.0:
Converged after 1892 iterations
Margin 7.0:
Converged after 2288 iterations
Margin 10.0:
Converged after 3044 iterations

As the margin for "Single Sample Perceptron with Margin" increases, the solution region compresses and hence the algorithm is taking more time(iterations) to converge.

Q- 4. RELAXATION ALGORITHM WITH MARGIN

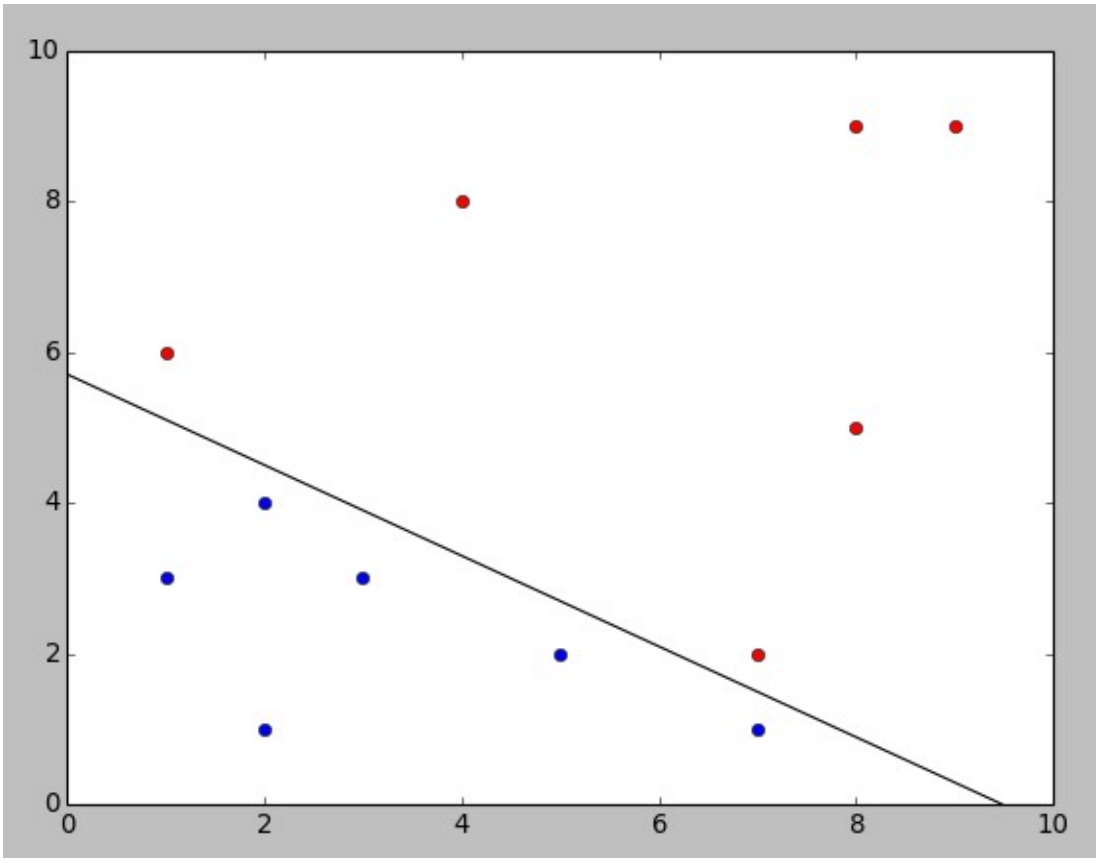
Algorithm:

begin initialize $\mathbf{a}, \eta(\cdot), k = 0$
  do $k \leftarrow k + 1$
    if $\mathbf{y}_k$ is misclassified **then** $\mathbf{a} \leftarrow \mathbf{a} + \eta(k)\frac{b - \mathbf{a}^t\mathbf{y}}{\|\mathbf{y}_k\|^2}\mathbf{y}_k$
    until all patterns properly classified
  return a
end

4. A) Plot the data points in a graph (e.g. Circle: class-1 and Cross: class-2) and also show the weight vector a learnt from all of the above algorithms in the same graph (labeling clearly to distinguish different solutions).

Assumptions:
Plotted Class 1 with Red Circles.
Plotted Class2 with Blue Circles.
The points on the solution vector is considered to be classified. Hence Accuracy here is 100%.
Learning Rate – 1.0
Margin - 2.0 (Initial Value of margin).
Initial weight vector – [0, 0, 1] (Can be taken as random as well).
Final weight vector - [2.43471529 4.04485064 -23.08870681] (Plotted as Black Line)

Sample Output:
Enter the type of algo to follow:
Enter 1.....Single-sample perceptron
Enter 2.....Single-sample perceptron with margin
Enter 3.....Relaxation algorithm with margin
Enter 4.....Widrow-Hoff or Least Mean Squared (LMS) Rule
Enter Choice
3
[2.43471529 4.04485064 -23.08870681]
Converged after 4038 iterations



4 B) Create a test set comprising three more data samples (y i) for each class and test your implementation by computing for the test samples the output (class label) predicted by the algorithm.

Test Set:
Test Set : [(6 , 4) ,(6, 6), (9 , 4), (0, 0), (0 ,-2) , (1, 1)]

Corresponding Class Labels :
[1,1,1,2,2,2]

Output:
[1 1 1 2 2 2]
Accuracy for the test data set is 100.0%

4 C) Run each of the above algorithms for various initial values of the weight vector, and comment on the dependence of convergence time (run-time) on initialization.

Assumptions:
The convergence time(run time) is measured in terms of the number of iterations the algorithm took to reach to the final solution vector.
The No of iterations are increasing proportionally wrt. the weight vector taken
Margin Value kept constant as : 1.0

Analysis on Various Initial Weight Vectors:

Weight Vector(Initial)	No of Iterations(Convergence Time)
[0, 0, 1]	Converged after 4020 iterations
[0, 1, 1]	Converged after 4073 iterations
[2, 2, 1]	Converged after 4282 iterations
[5, 5, 5]	Converged after 151 iterations
[5, 5, 10]	Converged after 102 iterations
[10, 15, 15]	Converged after 86 iterations
[50, 55, 55]	Converged after 89 iterations

Analysis : *As the weight vector is deviating more from the final solution vector, it is taking more time/no of iterations to converge to the final solution.*

4 D)Similarly explore the effect of adding different margins on the final solution as well as on the convergence (run-) time for algorithms (3) and (4).

Inital Weight Vector: [0, 0, 1]

Margin 1.0:
Converged after 4020 iterations
Margin 2.0:
Converged after 4038 iterations
Margin 3.0:
Converged after 4057 iterations
Margin 4.0:
Converged after 4061 iterations

As the margin for "Relaxation Algorithm with Margin" increases, the solution region compresses and hence the algorithm is taking more time(iterations) to converge.

Q- 5. WIDROW-HOFF OR LEAST MEAN SQUARED (LMS) RULE

Algorithm:

Choose:
$$\left.\begin{array}{l} \mathbf{a}(1) \qquad \text{arbitrary} \\ \mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)(b_k - \mathbf{a}(k)^t \mathbf{y}^k)\mathbf{y}^k, \end{array}\right\}$$

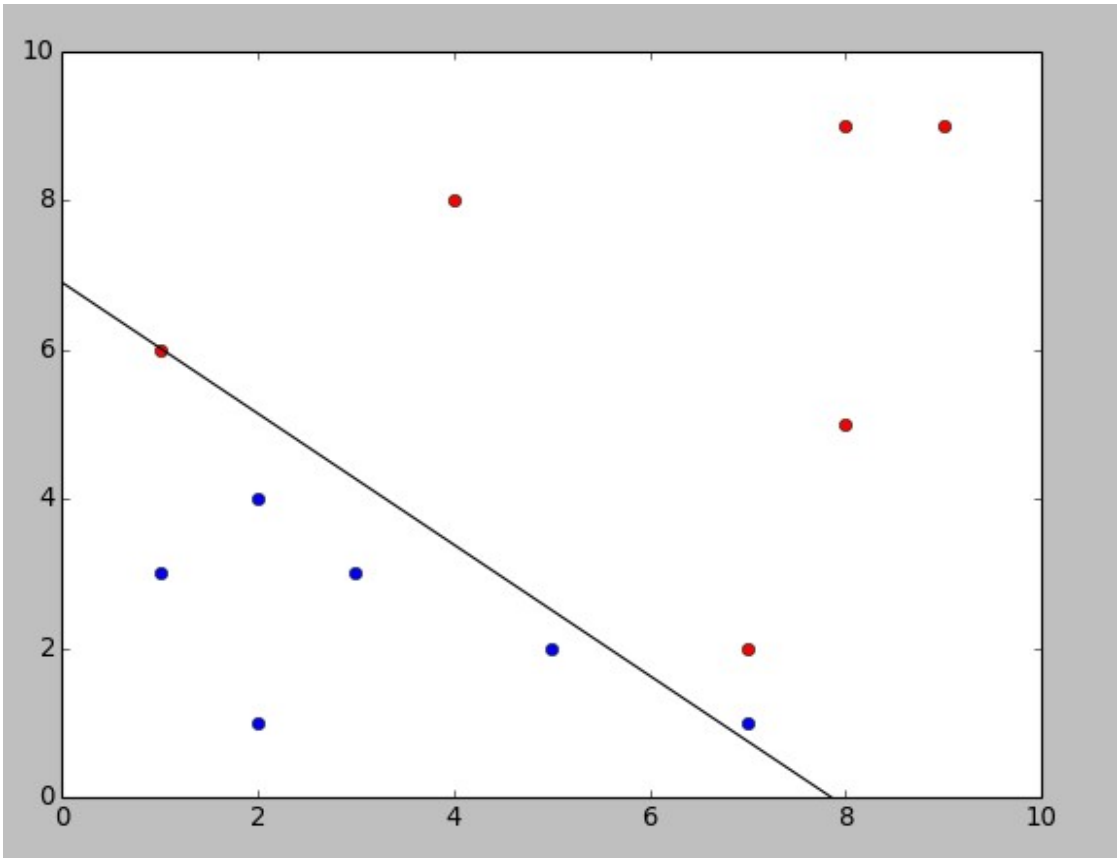
begin initialize $\mathbf{a}, \mathbf{b}, \text{criterion } \theta, \eta(\cdot), k = 0$
do $k \leftarrow k + 1$
$\mathbf{a} \leftarrow \mathbf{a} + \eta(k)(b_k - \mathbf{a}^t \mathbf{y}^k)\mathbf{y}^k$
until $\eta(k)(b_k - \mathbf{a}^t \mathbf{y}^k)\mathbf{y}^k < \theta$
return $\mathbf{a}$
end

Exact analysis of the behavior of the Widrow-Hoff rule in the deterministic case is rather complicated, and merely indicates that the sequence of weight vectors tends to converge to the desired solution. At first glance this descent algorithm appears to be essentially the same as the relaxation rule. The primary difference is that the relaxation rule is an error-correction rule, so that a t (k)y k does not equal b k , and thus the corrections never cease. Therefore, η(k) must decrease with k to obtain convergence, the choice of η(k) is η(1)/k.

5. A) Plot the data points in a graph (e.g. Circle: class-1 and Cross: class-2) and also show the weight vector a learnt from all of the above algorithms in the same graph (labeling clearly to distinguish different solutions).

Assumptions:
Plotted Class 1 with Red Circles.
Plotted Class2 with Blue Circles.
The points on the solution vector is considered to be classified.
Learning Rate – 0.01
Margin - 1 (Initial Value of margin).

Initial weight vector – [0.5,0.5,-1] (Can be any values in range -1 to +1).
Final weight vector -[0.06973139 0.18966708 -1.10174966] (Plotted as Black Line)



5 B) Create a test set comprising three more data samples (y i) for each class and test your implementation by computing for the test samples the output (class label) predicted by the algorithm.

Test Set:
[(6 , 4) ,(6, 6), (9 , 4), (0, 0), (0 ,-2) , (1, 1)]

Corresponding Class Labels :
[1,1,1,2,2,2]

Output:
[1 2 1 2 2 2]
Accuracy for the test data set is 83.3333333333%

5 C) Run each of the above algorithms for various initial values of the weight vector, and comment on the dependence of convergence time (run-time) on initialization.

Assumptions:
The convergence time(run time) is measured in terms of the number of iterations the algorithm took to reach to the final solution vector.
The No of iterations are increasing proportionally wrt. the weight vector taken

Analysis on Various Initial Weight Vectors:

Analysis : *As the weight vector is deviating more from the final solution vector, it is taking more time/no of iterations to converge to the final solution.*

5 D)Similarly explore the effect of adding different margins on the final solution as well as on the convergence (run-) time for algorithms (3) and (4).

Margin : 1.0
Accuracy for the test data set is 88.8888888889%

Margin 2.0
Accuracy for the test data set is 77.7777777778%

Margin 3.0
Accuracy for the test data set is 66.6666666667%

Margin 4.0
Accuracy for the test data set is 61.1111111111%

Margin 5.0
Accuracy for the test data set is 66.6666666667%

As the margin is increased, the minimum distance of the point from the solution is increased and hence the accuracy is decreased.

Comparison table listing test set accuracies of each of the above algorithms:

	Single-sample perceptron	Single-sample perceptron with margin	Relaxation algorithm with margin	Widrow-Hoff or Least Mean Squared (LMS) Rule
ACCURACY	83.3333333333%	100%	100%	83.3333333333%

** The Accuracy of the algorithm depends on the test set taken. The test set taken here is closer to the solution vector hence tends to missclassify.*

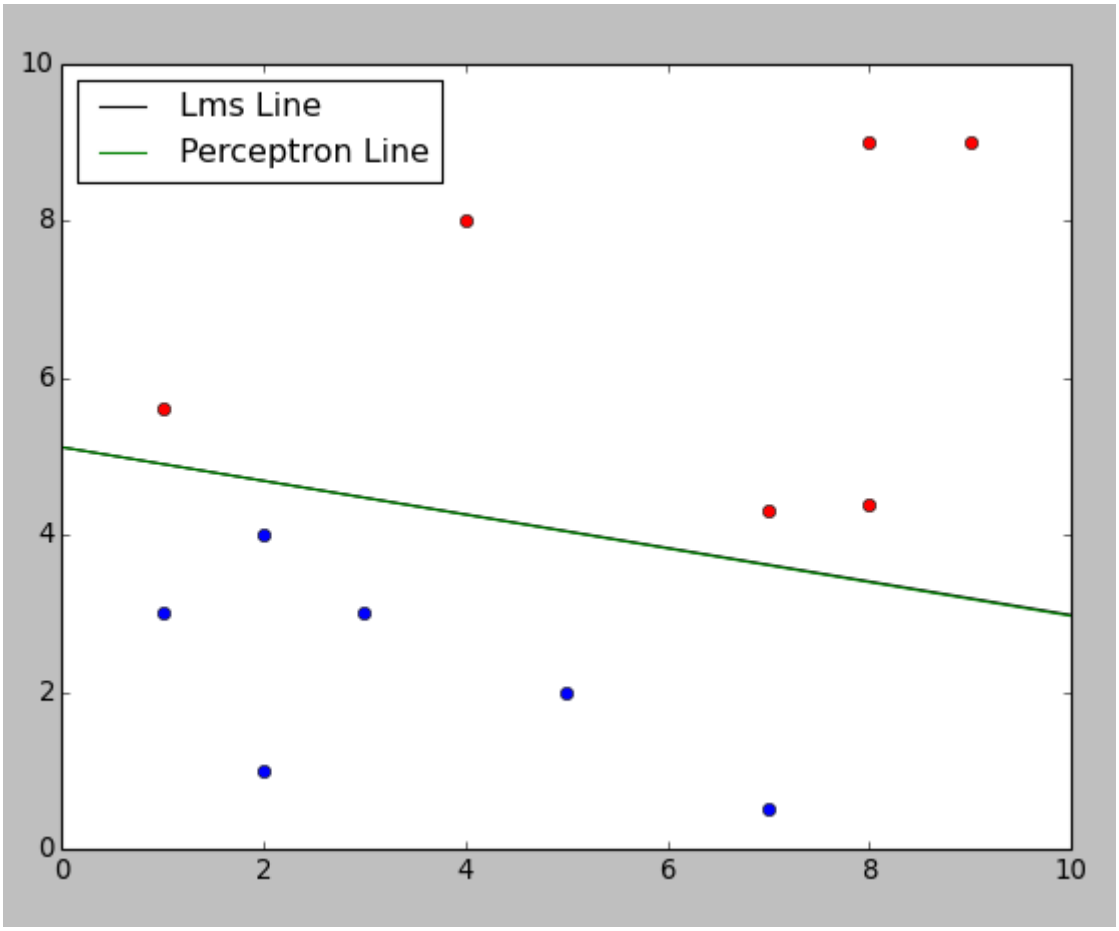
Q- 6. Remember the discussion on LMS and Perceptron solutions being different in some cases – can you experiment and construct a training set that makes the solution of LMS rule differ from those of the other. Of course, if the dataset given above already yields different solutions, can you adjust the dataset so that the solutions align!

Sol - The solution was different for the given dataset for LMS and Perceptron. Points were missclassified in LMS for the given datasets and the weight vector was also plotted in a different way. The slope of the two weight vectors were different that was required to be changed.

Changes:

X= [(1, 6), (7, 2), (8, 9), (9, 9), (4, 8), (8, 5), (2, 1), (3, 3), (2, 4), (7, 1), (1, 3), (5, 2)]

X'= [(1, 5.6), (7, 4.3), (8, 9), (9, 9), (4, 8), (8, 4.4), (2, 1), (3, 3), (2, 4), (7, 0.5), (1, 3), (5, 2)] - X' is the modified dataset.



The points werechanged step by step so the weight vectors of the LMS Alogirthm & Perceptron Algo coincide/ Align.

There are two lines (LMS – Black) and (Perceptron – Green) which are coincided and appears to be a single line in the above plot.

Q- 7. Modify the dataset such that it becomes linearly non-separable (clearly list the changes in the report). Now run algorithms (4) and (5) with suitable stopping criteria. Plot the data points in a graph (e.g. Circle: class- w 1 and Cross: class- w2) and also show the weight vector a learnt from these two algorithms in the same graph (labeling clearly to distinguish different solutions) and comment on the nature of the solution found in each case.

Sol-

Changes to make dataset linearly non-separable:
Exchange some coordinates of class 1 with class 2.
(1, 6) is exchanged with (2,1)
(8, 9) is exchanged with (2, 4)
(8, 5) is exchanged with (5, 2)

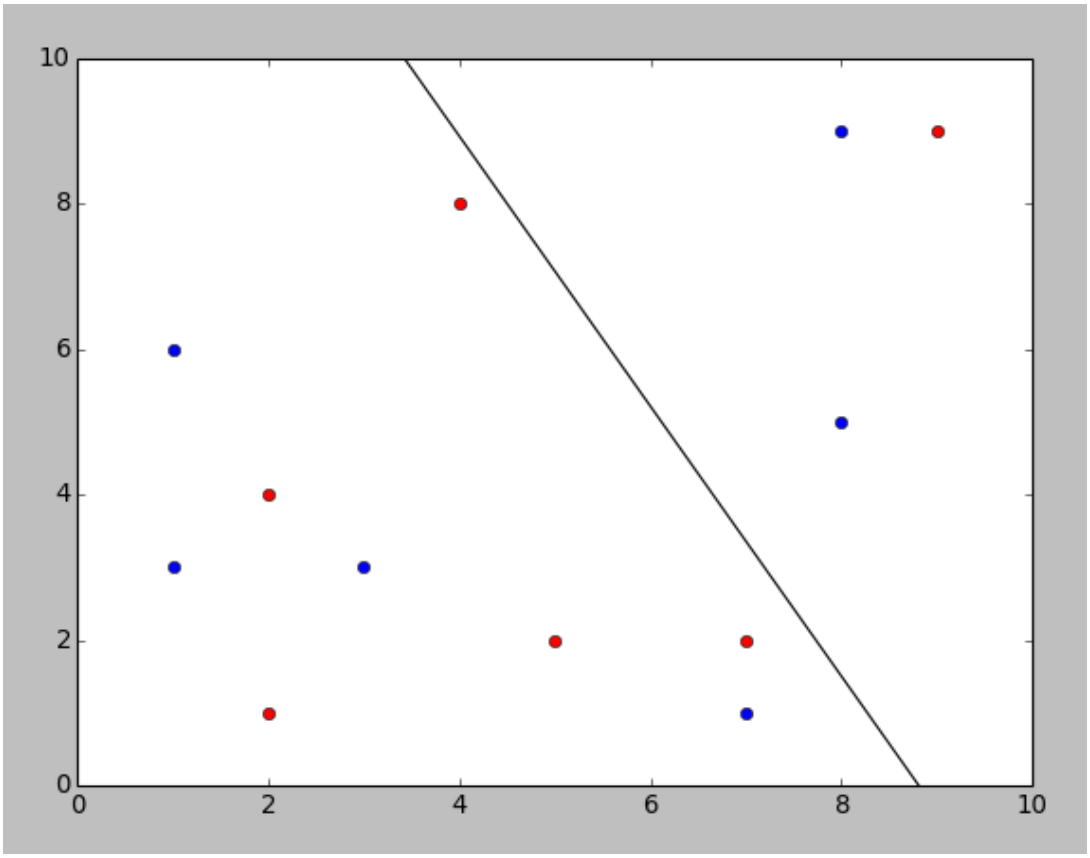
Now, coordinated of class 1 and class 2 lies together in the graph.
X' is the modified datasets.

X= [(1, 6), (7, 2), (8, 9), (9, 9), (4, 8), (8, 5), (2, 1), (3, 3), (2, 4), (7, 1), (1, 3), (5, 2)]
X'= [(2, 1), (7, 2), (2, 4), (9, 9), (4, 8), (5, 2),(1, 6), (3, 3), (8, 9), (7, 1), (1, 3), (8, 5)]

Plotting Assumptions:
Plotted Class 1 with Red Circles.
Plotted Class2 with Blue Circles.

Observation:
Algorithm(4): Relaxation algorithm with margin
Perceptron Doesnt converge for linearly non-separable classes.
Hence the loop is going in the infinite loop, as some misclassified samples will always be left, and we wont be able to plot the solution vector with Relaxation Algorithm of linearly Non-Separable Classes.

Algorithm(5): Widrow-Hoff or Least Mean Squared (LMS) Rule
Inital Weight=[0.5,0.5,-1]
Margin = 1.0
The algorithm converges with a solution vector that doesnt separate the classes.


10
8
6
4
2
0
0
2
4
6
8
10