STATISTICAL METHODS IN AI
ASSIGNMENT-2:
Linear Discriminant Functions

Name – Kalpish Singhal

M.Tech CSE (201505513)

Dated : 2 February, 2016

Preamble:

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.

Contents:

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).

ASSUMPTIONS

1) We keep on feeding the pattern's $X(n)$. $n = 1, 2, \ldots$
2) $W(0)$, weight vector is considered to be initialized by
3) Lineary seprability of given classes.
4) $n(K)$ is constant - there is fixed increment.
5) Initially we assume when we feed $x_1, x_2 \ldots$ 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
→ If some value of exist's that give's correct classification, then we can say that system is covergr

Fixed increment rule:-

$$a(1) - \text{arbitary} \quad a(K+1) = a(K) + y^k, \; K \geq 1 \quad (1)$$

If a_2 is any solution vector, then

$$\|a_2(K+1) - a_2\| \leq \|a(K) - a_2(K)\|$$

Since a_2 is a solution vector.

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

∴ we can add a scale factor to (1) & rewrite it as:

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

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

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

Let β be the max pattern vector length.

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

& 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 $a(K)$ & αa_2 is reduced by atleast β^2 after each correction/itteration

So, After K itteration:-

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

the Squared distance cannot be -ve.

So, at most K_0 correction's & $K_0 = \|a(1) - \alpha a_2\|^2 / \beta^2$

Hence K_0 is a bound on Number of correction's & there will alway's be finite Number of such itteration's If Sample is 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.

w1 = [(1; 6); (7; 2); (8; 9); (9; 9); (4; 8); (8; 5)]

w2 = [(2; 1); (3; 3); (2; 4); (7; 1); (1; 3); (5; 2)]

ALGORITHMS:

Single-sample perceptron

fixed- y1, y3, y1, y2, y2, ... With this understanding, the fixed-increment rule for generating increment rule a sequence of weight vectors can be written as a(1) arbitrary a(k + 1) = a(k) + yk k ≥ 1 (20) where at (k)yk ≤ 0 for all k. If we let n denote the total number of patterns, the algorithm is: (Fixed-increment single-sample Perceptron)

begin initialize a, k = 0
    do k ← (k + 1)mod n
        if yk is misclassified by a **then** a ← a − yk
    until all patterns properlyclassified
    return a 6
end

Single-sample perceptron with margin

$$\left.\begin{array}{ll}\mathbf{a}(1) & \text{arbitrary} \\ \mathbf{a}(k+1)=\mathbf{a}(k)+\eta(k)\mathbf{y}^k & k \geq 1,\end{array}\right\}$$

where now $\mathbf{a}^t(k)\mathbf{y}^k \leq b$ for all k. Thus for n patterns, our algorithm is:

Algorithm 5 (Variable increment Perceptron with margin)

begin initialize $\mathbf{a}$, criterion θ, margin b, $\eta(\cdot)$, $k=0$
    do $k \leftarrow k+1$
        if $\mathbf{a}^t\mathbf{y}_k + b < 0$ **then** $\mathbf{a} \leftarrow \mathbf{a} - \eta(k)\mathbf{y}_k$
    until $\mathbf{a}^t\mathbf{y}_k + b \leq 0$ for all k
  return $\mathbf{a}$
end

Relaxation algorithm with margin

Algorithm 9 (Single-sample relaxation with margin)

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 $\mathbf{a}$
end

This algorithm is known as the *single-sample relaxation rule with margin*, and it has a simple geometrical interpretation. The quantity

Widrow-Hoff or Least Mean Squared (LMS) Rule

LMS RULE tially and using the *Widrow-Hoff* or *LMS rule* (least-mean-squared):

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

or in algorithm form:

Algorithm 10 (LMS)

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

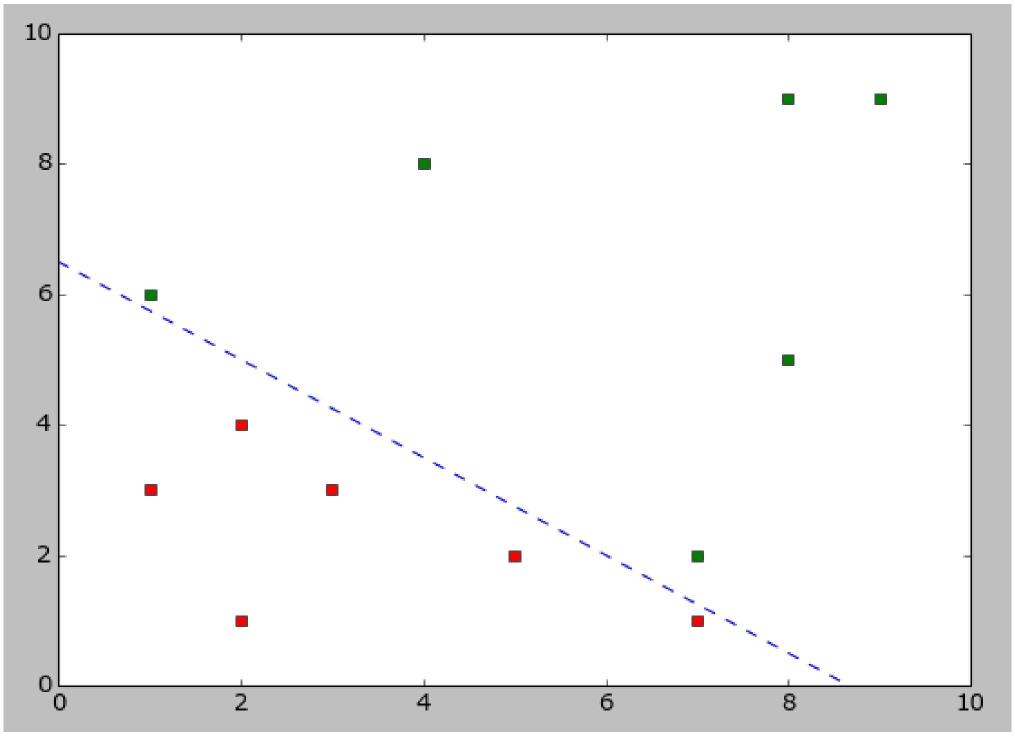
<u>**IMPLEMENTATION:**</u>

<u>Ques 2 SINGLE SAMPLE PERCEPTRON</u>

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 Squares and Class2 with Green Sqaures.
Learning Rate – 1.0 With Margin - 0.0
Initial weight vector –Random/ [0, 0, 1]
Final weight vector - [3. 4. -26.] (Plotted as Blue Dotted Line)

Output:
Enter your choice:
i) 1 for Single-sample perceptron
ii)2 for Single-sample perceptron with margin
iii)3 for Relaxation algorithm with margin
iv)4 for Widrow-Hoff or Least Mean Squared (LMS) Rule
Enter Choice 1
Number of iterations for conversion are 392.
weight is: [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 : [(7 , 5) ,(6, 7), (8 , 3), (1, 0), (0 ,1) ,(1, 1)]
Class : [1,1,1,2,2,2]

Prediction:
[1, 1, 1, 2, 2, 2]
Accuracy is: 100%

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.
PRATICAL:-
Weight vector : [1 1 1]
iterations: 428.
Weight vector : [2 1 2]
iterations: 801.
Weight vector : [3 2 1]
iterations: 260.
Weight vector : [3 -1 .05]
iterations: 416.

Weight vector : [3 4 -5]
iterations: 344.

Analysis : As the weight vector is ariving nearer to final solution vector, it is taking less time/no of iterations to converge to the final solution.

Q- 3. SINGLE-SAMPLE PERCEPTRON WITH MARGIN

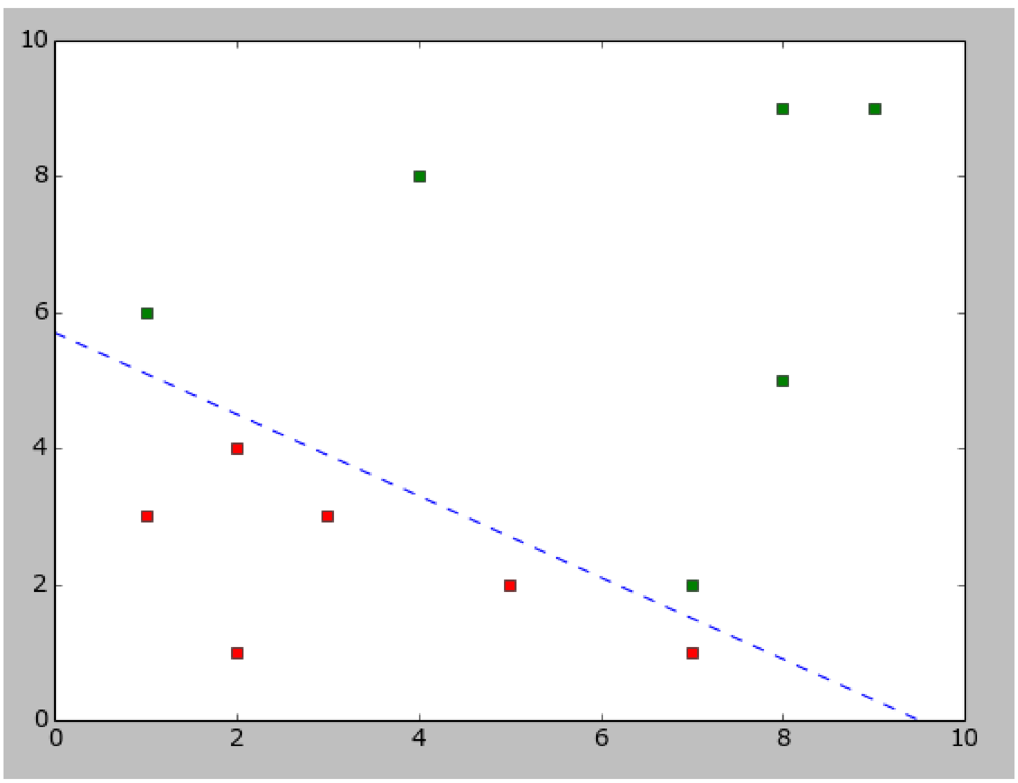
3. A) Plot the data points in a graph (e.g. Circle: class-1 and Cross: class-2) and also show the weight vector a learnt from all of the above algorithms in the same graph (labeling clearly to distinguish different solutions).

Assumptions:

Plotted Class 1 with Red Squares and Class2 with Green Sqaures.
Learning Rate – 1.0 With Margin - 2.0
Initial weight vector –Random/ [0, 0, 1]
Final weight vector - [6. 10. -57.] (Plotted as Blue Dotted Line)

Output:

Enter your choice:
i) 1 for Single-sample perceptron
ii)2 for Single-sample perceptron with margin
iii)3 for Relaxation algorithm with margin
iv)4 for Widrow-Hoff or Least Mean Squared (LMS) Rule
Enter Choice 2
Number of iterations for conversion are 1281.
weight is: [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 : [(7 , 5) ,(6, 7), (8 , 3), (1, 0), (0 ,1) ,(1, 1)]
Class : [1,1,1,2,2,2]

Prediction:
[1, 1, 1, 2, 2, 2]
Accuracy is: 100%

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.
PRATICAL:-
Weight vector : [1 1 1]
iterations:1245.
Weight vector : [3 1 2]
iterations:.1269.
Weight vector : [2 4 -7]
iterations:1173
Weight vector : [2 5 -15]
iterations:1077.
Weight vector : [4 5 -25]
iterations:945.
Weight vector : [5 6 -30]
iterations:849.
***Analysis* :** As the weight vector is ariving nearer to final solution vector, it is taking less 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]
Learning Rate: 1.0

Margin Value	**No of Iterations Taken**
1.0	705 Iterations

2.0	1245 Iterations
5.0	1857 Iterations
8.0	2240 Iterations
10.0	3021 Iterations

- The Behaviour of the margin for the no. of iterations depends on the input data set.

For the Given Data Set :

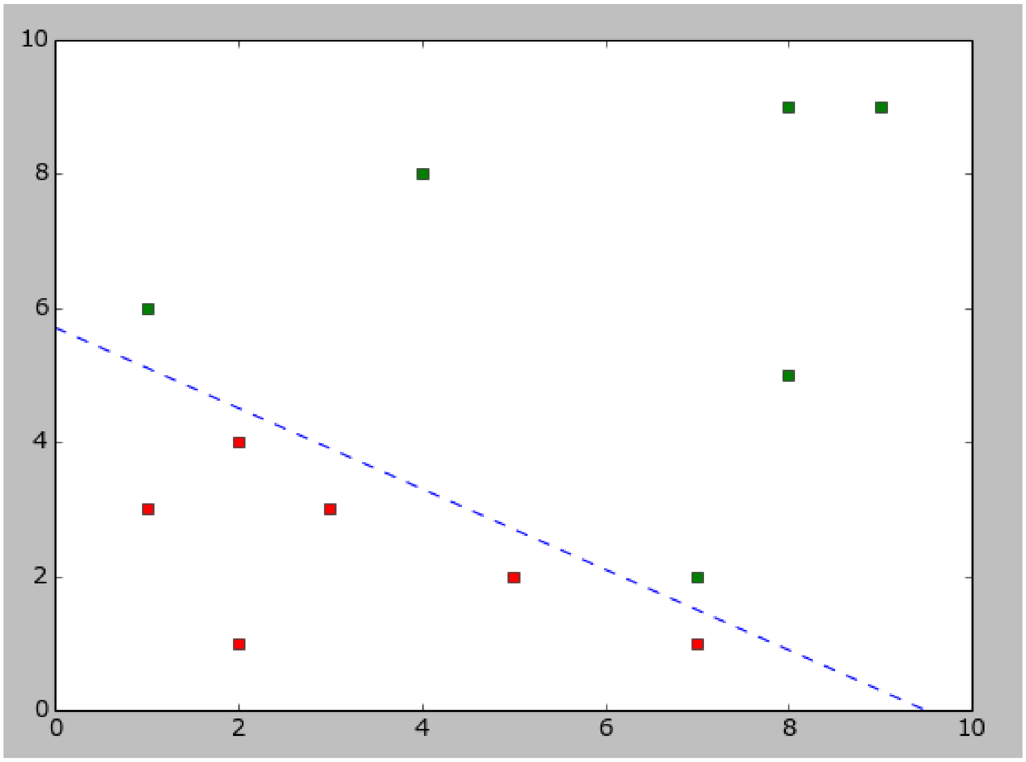
- No. of iterations are increasing as the margin value increases.
- As the margin value is increasing, the solution region compresses and hence the algo takes more time to converge i.e. more number of iterations.
- This is not the case always. A better solution can be found in less number of iterations as well with increased margin. It depends on the convergence time and behaviour w.r.t. to the margin.

Q- 4. RELAXATION ALGORITHM WITH MARGIN

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 Squares and Class2 with Green Sqaures.
Learning Rate – 1.0 With Margin - 2.0
Initial weight vector –Random/ [0, 0, 1]

Output:
Enter your choice:
i) 1 for Single-sample perceptron
ii)2 for Single-sample perceptron with margin
iii)3 for Relaxation algorithm with margin
iv)4 for Widrow-Hoff or Least Mean Squared (LMS) Rule
Enter Choice 3
Number of iterations for conversion are 4047.
weight is: [2.43176221 4.04385168 -23.08898237]



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 : [(7 , 5) ,(6, 7), (8 , 3), (1, 0), (0 ,1) ,(1, 1)]
Class : [1,1,1,2,2,2]

Prediction:
[1, 1, 1, 2, 2, 2]
Accuracy is: 100%

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.

PRATICAL:-
Weight vector : [1 1 1]
iterations:4049.
Weight vector : [10 10 7]
iterations:4035.
Weight vector : [11 7 -100]
iterations:4514.
Weight vector : [3 25 -10]

iterations:4056.
Weight vector : [16 15 -60]
iterations:4028.
Weight vector : [12 17 -150]
iterations:2342.

Analysis **:**

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]
Learning Rate: 1.0

Margin Value	**No of Iterations Taken**
1.0	4011 Iterations
2.0	4047 Iterations
3.0	4053 Iterations
5.0	4058 Iterations

- The Behaviour of the margin for the no. of iterations depends on the input data set.

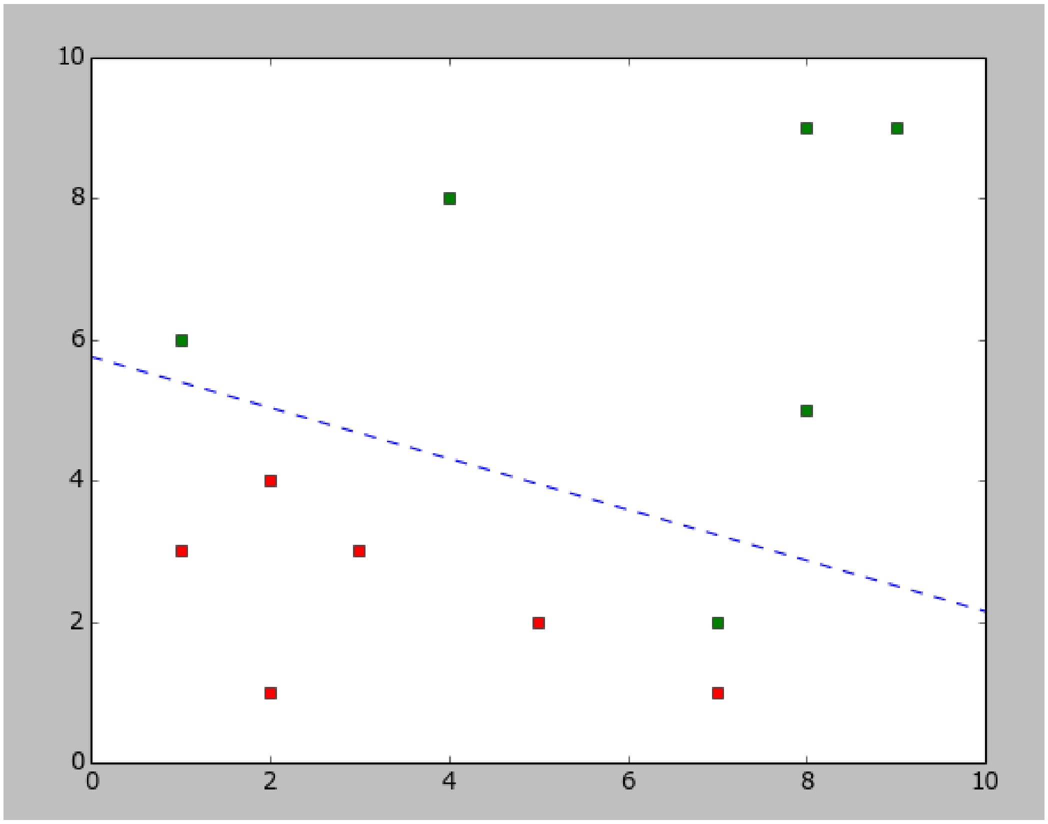
  For the Given Data Set :
- No. of iterations are increasing as the margin value increases.
- As the margin value is increasing, the solution region compresses and hence the algo takes more time to converge i.e. more number of iterations.
- This is not the case always. A better solution can be found in less number of iterations as well with increased margin. It depends on the convergence time and behaviour w.r.t. to the margin.

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

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 Squares and Class2 with Green Sqaures.
Learning Rate – 0.01 With Margin - 1.0

Initial weight vector –Random/ [0.5,0.5,-1]
Final weight vector - [0.0712864   0.19734848 -1.13596597] (Plotted as blue Dotted Line)



Output:
Enter your choice:
i) 1 for Single-sample perceptron
ii)2 for Single-sample perceptron with margin
iii)3 for Relaxation algorithm with margin
iv)4 for Widrow-Hoff or Least Mean Squared (LMS) Rule
Enter Choice 4
weight is: [0.0712864   0.19734848 -1.13596597]

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:
Test Set : [(7 , 5) ,(6, 7), (8 , 3), (1, 0), (0 ,1) ,(1, 1)]
Class : [1,1,1,2,2,2]

Prediction:
[1, 1, 1, 2, 2, 2]
Accuracy is: 100%

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 : 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.

Comparison table listing Taken 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	100.00%	100%	100%	100%

<u>CODE:</u>

```
from numpy import *
from pylab import *
from math import *
import matplotlib.pyplot as plt
w12_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)]
w12_y=[1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2]
X_NON_seprable=[(1, 6),(3, 3),(5, 2), (9, 9), (4, 8), (8, 5), (2, 7),  (7, 2), (2, 4), (7, 1), (1, 3), (8, 9) ]
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)]
def plot_bndry_double(weight_ss,weight,passing):
	X,Y,training_set=normal_train_set(passing)
```

```
        x_points=X[:,0]
        length=len(x_points)
        y_points =X[:,1]
        plt.axis([0,10,0,10])
        plt.plot(x_points[:length/2],y_points[:length/2],'gs');
        plt.plot(x_points[length/2:],y_points[length/2:],'rs');
        x_cordinate=-(float(weight[2]))/(float(weight[0]))
        y_cordinate=-(float(weight[2]))/(float(weight[1]))
        Lms,=plt.plot([0, x_cordinate],[y_cordinate, 0],'red',label='Lms')
        x_cordinate=-(float(weight_ss[2]))/(float(weight_ss[0]))
        y_cordinate=-(float(weight_ss[2]))/(float(weight_ss[1]))
        Perceptron_Line,=plt.plot([0,x_cordinate],
[y_cordinate,0],'y--',label='Perceptron')
        plt.legend([Lms, Perceptron_Line],['Lms', 'Perceptron'])
        plt.show()
        pass

def plot_bndry(weight,passing):
        X,Y,training_set=normal_train_set(passing)
        x_points=X[:,0]
        length=len(x_points)
        y_points =X[:,1]
        plt.axis([0,10,0,10])
        plt.plot(x_points[:length/2],y_points[:length/2],'gs');
        plt.plot(x_points[length/2:],y_points[length/2:],'rs');
        x_cordinate=-(float(weight[2]))/(float(weight[0]))
        y_cordinate=-(float(weight[2]))/(float(weight[1]))
        plt.plot([0, x_cordinate],[y_cordinate, 0],'b--')
        plt.show()
        pass
def compute_accuracy(weight):
        pred_list = []
        X_test = [(7 , 5) ,(6, 7), (8 , 3), (1, 0), (0 ,1) ,(1, 1) ]
        Y_test = [ 1, 1, 1, 2, 2, 2]
        X_test_len=len(X_test)
        X_test=hstack((X_test, ones((X_test_len, 1))))

        for i in range(0,len(X_test)):
                dot_value = dot(X_test[i], weight)
                if dot_value>0:
```

```
                pred_list.append(1)
            elif dot_value<0:
                pred_list.append(2)
            elif dot_value == 0:
                pass
        print pred_list
        count=0
        for i in range(len(pred_list)):
            if pred_list[i] == Y_test[i]:
                count+=1

        length = len(pred_list)
        accuracy = (count/length)*100
        return accuracy
        pass

def relaxation_algo_with_margin(margin,learning_rate,passing):
        X,Y,training_set=normal_train_set(passing)
        n=len(training_set)
        i=count=0
        k=-1
        weight=[1,1,1]
        while i!=n:
            k=(k+1)%n
            dot_p=dot(training_set[k],weight)
            if margin>=dot_p:
                temp=float((float(margin-
dot_p)/float(dot(training_set[k],training_set[k])))*2)
                weight=weight + (learning_rate * dot(temp,training_set[k]))
                i=0
            else:
                i=i+1
                count=count+1
        plot_bndry(weight,passing)
        print "Number of iterations for conversion are %d." % count
        print "weight is: %s" % weight
        return weight
        pass
def normal_train_set(passing):
        if passing==1:
```

```
        X=asarray(w12_x)
        Y=asarray(w12_y)
    elif passing==2:
        X=asarray(Xnew)
        Y=asarray(w12_y)
        pass
    elif passing==3:
        X=asarray(X_NON_seprable)
        Y=asarray(w12_y)
    LEN_X=len(X)
    train_set=hstack((X, ones((LEN_X, 1))))
    convert_truth= Y==unique(Y)[1]
    train_set[convert_truth]=-train_set[convert_truth]
    return X,Y,train_set
    pass
def perceptron_single_sample(margin,learning_rate,passing):
    X,Y,training_set=normal_train_set(passing)
    n=len(training_set)
    weight=[1,1,1]
    i=count=0
    k=-1
    while i!=n:
        count=count+1
        k = (k+1)%n
        if margin>=dot(training_set[k],weight):
            weight=training_set[k]* learning_rate + weight
            i=0
        else:
            i=i+1
    if passing==1:
        plot_bndry(weight,passing)
    print "Number of iterations for conversion are %d." % count
    print "weight is: %s" % weight
    return weight
    pass


def least_mean_square(margin,learning_rate,passing):
    X,Y,training_set=normal_train_set(passing)
```

```
	count=0
	k=-1
	weight=[0.5,0.5,-1]
	var=1
	while var == 1:
		count=count+1
		flag=1
		k=(k+1)%len(training_set)
		nk=1.0/2000.0
		temp=dot((nk*                    (margin                    -
dot(training_set[k],weight))),training_set[k])
		if sqrt(dot(temp,temp))<margin:
			flag=0
		weight=sum([weight,temp],axis=0)
		if(flag==1 or count==2000):
			break;
	print "weight is: %s" % weight
	if passing==1 or passing ==3 :
		plot_bndry(weight,passing)
	elif passing==2:
		weight_ss=perceptron_single_sample(margin,learning_rate,passing)
		plot_bndry_double(weight_ss,weight,passing)

	print "Number of iterations for conversion are %d." % count
	return weight
	#Obtained the values as required (Augmented and negated)
	pass

def main():
	print 'Enter your choice: \ni) 1 for Single-sample perceptron \nii)2 for
Single-sample perceptron with margin \niii)3 for Relaxation algorithm with
margin  \niv)4 for Widrow-Hoff or Least Mean Squared (LMS) Rule.\nv)5 LMS
and perceptron solution align'
	input_case=input("Enter Choice\n")
	if input_case == 1:
		m=0.0
		lern_rate = 1.0
		weight=perceptron_single_sample(m,lern_rate,1)
		accuracy = compute_accuracy(weight)
		print "Accuracy is: %d" % accuracy + "%"
```

```
            pass

        if input_case == 2:
            m=2
            lern_rate = 1.0
            weight=perceptron_single_sample(m,lern_rate,1)
            accuracy = compute_accuracy(weight)
            print "Accuracy is: %d" % accuracy + "%"
            pass

        if input_case == 3:
            m=15
            lern_rate=1.0
            weight=relaxation_algo_with_margin(m,lern_rate,1)
            accuracy = compute_accuracy(weight)
            print "Accuracy is: %d" % accuracy + "%"
            pass

        if input_case == 4:
            m=1
            lern_rate=0.01
            weight=least_mean_square(m,lern_rate,1)
            accuracy = compute_accuracy(weight)
            print "Accuracy is: %d" % accuracy + "%"
            print 'Non Separable Data Plotting'
            weights = least_mean_square(m,lern_rate,3)
            pass
        if input_case == 5:
            lern_rate =0.01
            m = 1.0
            weight=least_mean_square(m,lern_rate,2)
        if input_case>5 or input_case<1:
            print "wrong input"


main()
```

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. So step by step Points were changed in initial X such that it got alligend and solution finally have same ovelapping lines as shown in the figure Red line of LMS and yellow dotted line of *Perceptron solution align.*

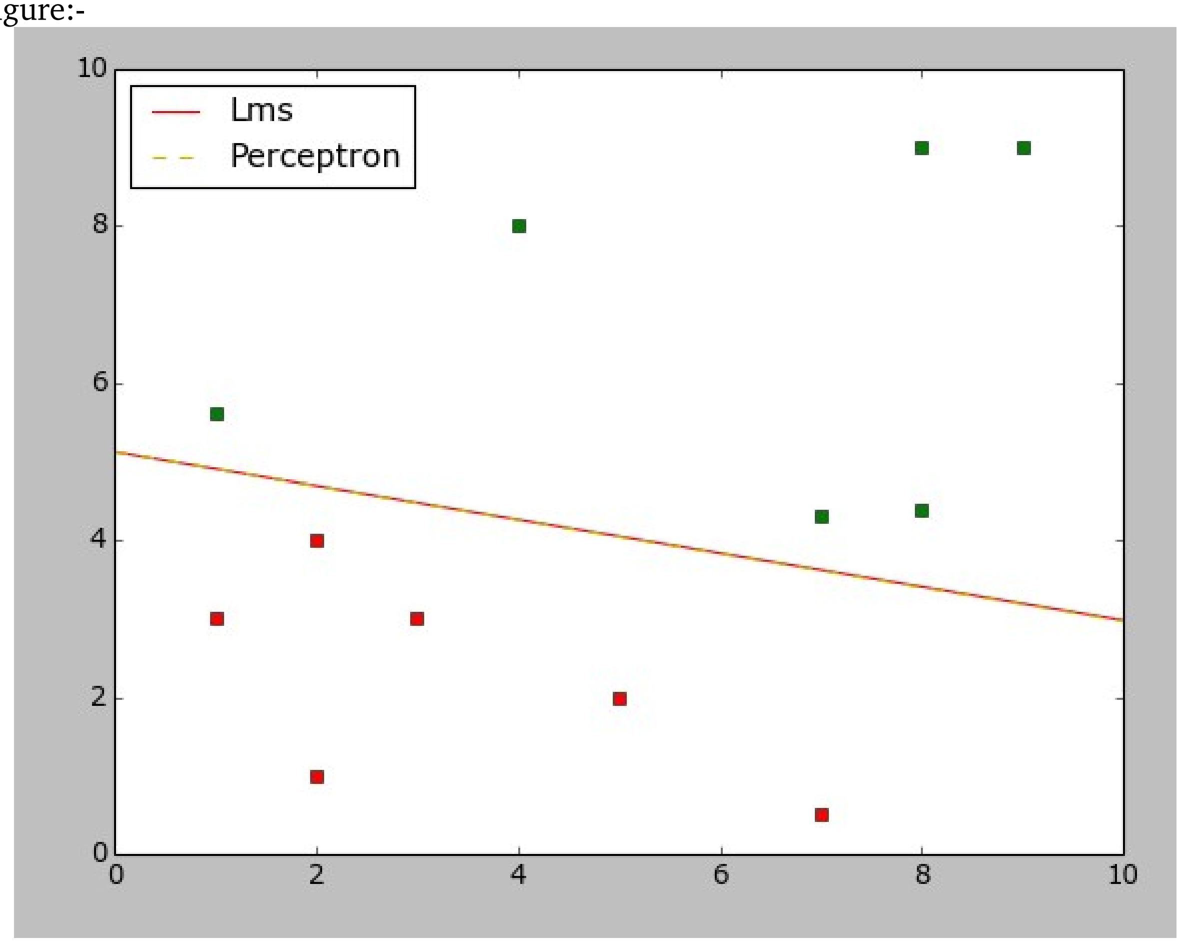
Changes:

Changed the second Coordinate (7,2) of class 1 (Which was missclassified) to (7,6).

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)]

Sol figure:-



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 samples of class 1 with class 2.

- ✔ (7, 2) is exchanged with (3, 3)
- ✔ (5,2) is exchanged with (8, 9)
- ✔ ((2, 1) is changed with (2, 7)
- ✔ Coordinates of both the classes class 1 and 2 lies together in graph.

X_NON_seprable is the new dataset taken for it .

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

<u>**Algorithm: Relaxation algorithm with margin**</u>

- ➢ Perceptron Doesnt converge for linearly non-separable classes.
- ➢ The loop is going in the infinite loop, as some misclassified samples will always be left.
- ➢ Plotting of solution vector for linearly non-separable classes is not possible in this case.

<u>**Algorithm: Widrow-Hoff or Least Mean Squared (LMS) Rule**</u>

- ➢ The algorithm converges with a solution vector that doesnt separate the classes.

Plotting:
Plotted Class 1 with Red Square.
Plotted Class2 with Green Square.