B. Prabhakar

==========

201505618

==========

Assignment II

Question 1:

Intuitive Explanation on Convergence :-

The initial vector be α_0, as the hyperplane (straight line in this case) is misclassifying a couple of datapoints, as we'll be updating the weight vector using the perceptron update rule.

Update rule :- (assuming $\eta = 0.5$)

$$\alpha_2 = \alpha_0 + \eta \gamma_1$$

The resulting weight vector gives a decision boundary that correctly classifies the vectors γ_1 data set.

The updated α_1 correctly classifies the misclassified γ_1.

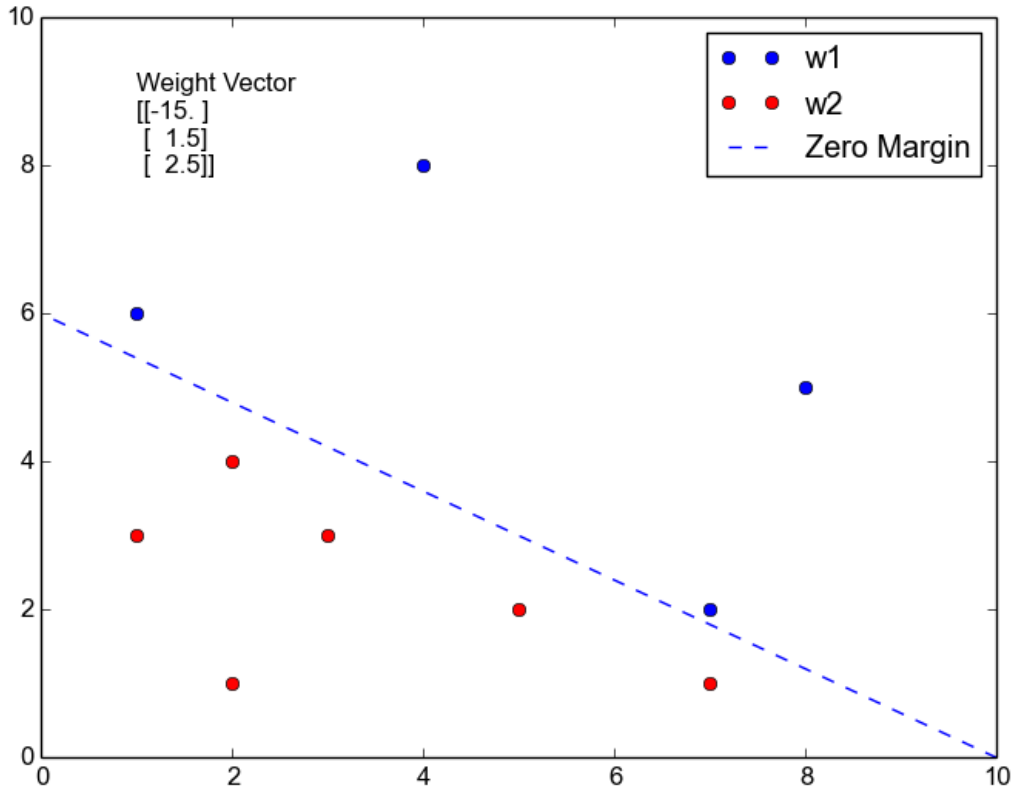
In this manner, the perceptron algo keeps continuing until all the data set gets classified correctly.

For Assuming, the data set is linearly-separable, the algorithm always converges after finding a linear decision boundary that correctly classifies the data-set.

Question 2:

Single-sample Perceptron

prabhakar@Code$ *python Algo_implemented.py SingleSamplePerceptron*



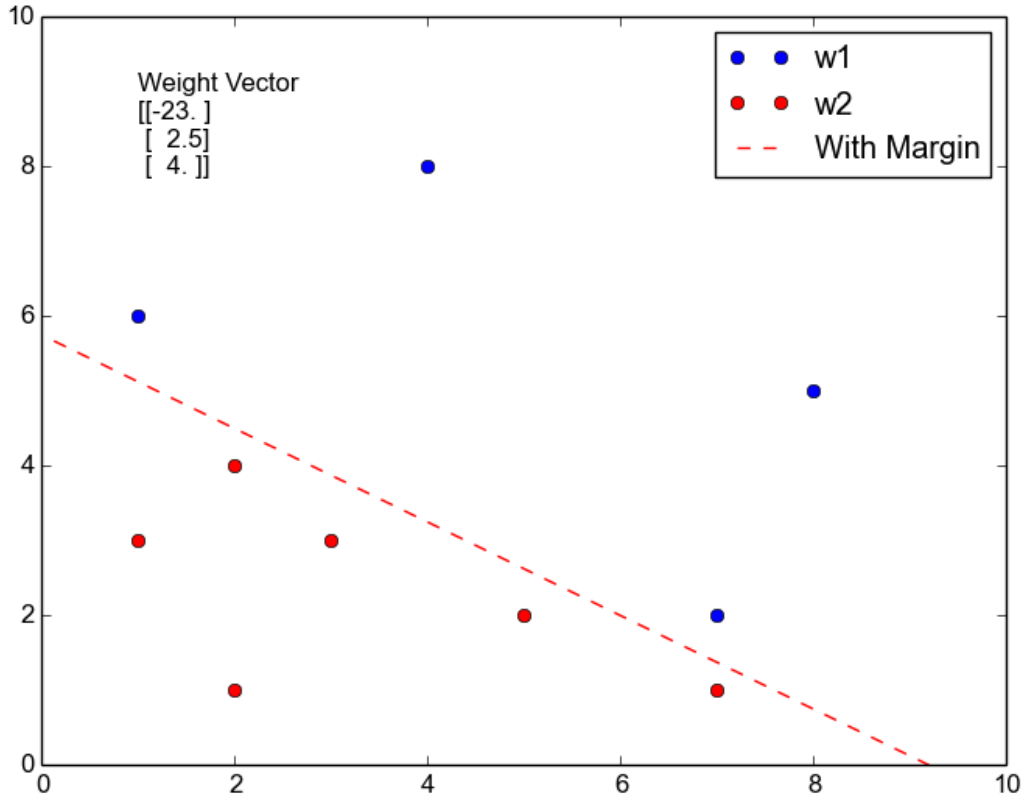
Weight Vector:

$$\begin{vmatrix} -15 \\ 1.5 \\ 2.5 \end{vmatrix}$$

Question 3:

Single-sample Perceptron with Margin

prabhakar@Code$ python Algo_implemented.py SinglesamplePerceptronWithMargin



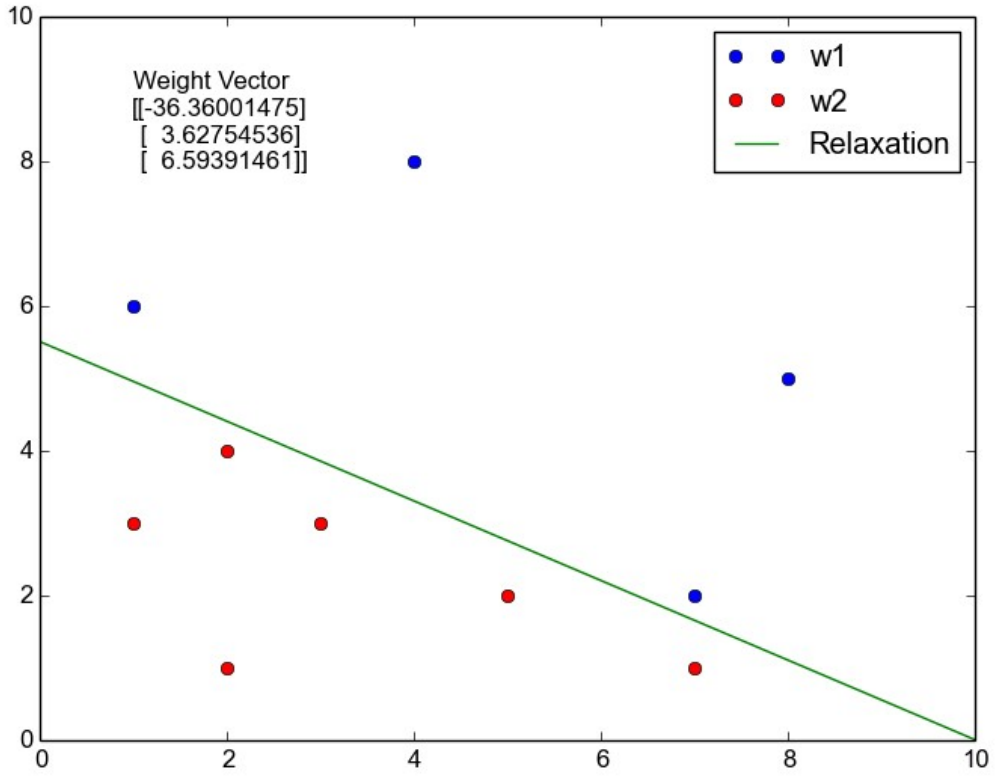
Weight Vector:

$$\begin{vmatrix} -23 \\ 2.5 \\ 4 \end{vmatrix}$$

Question 4:

Relaxation algorithm with margin

prabhakar@Code$ python Algo_implemented.py Relaxation



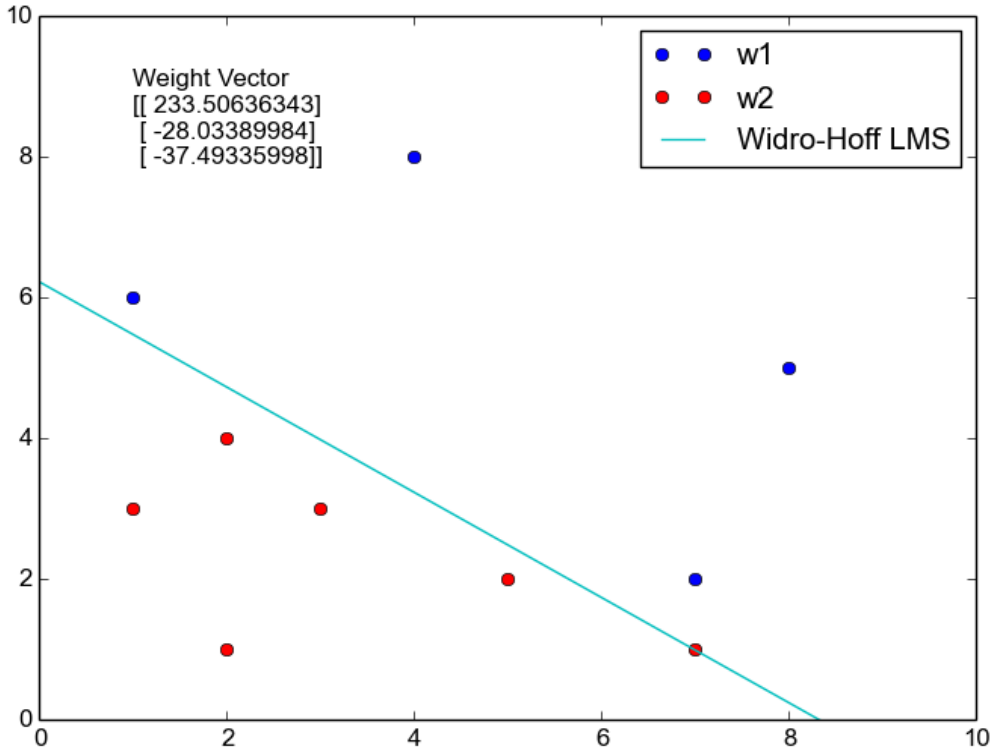
Weight Vector:

$$\begin{vmatrix} -36.36001475 \\ 3.62754536 \\ 6.59391461 \end{vmatrix}$$

Question 5:

Widrow-Hoff or Least Mean Squared (LMS) Rule

prabhakar@Code$ python Algo_implemented.py LMS



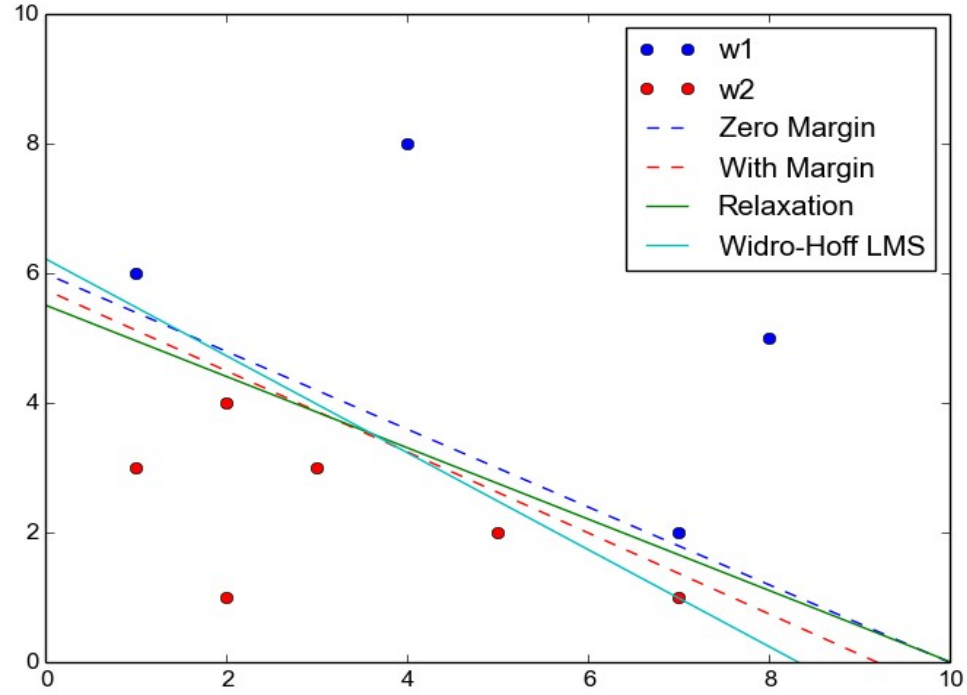
Weight Vector:

$$\begin{vmatrix} 233.50636343 \\ -28.03389984 \\ -37.49335998 \end{vmatrix}$$

Question A:

Widrow-Hoff or Least Mean Squared (LMS) Rule

prabhakar@Code$ python Algo_implemented.py all



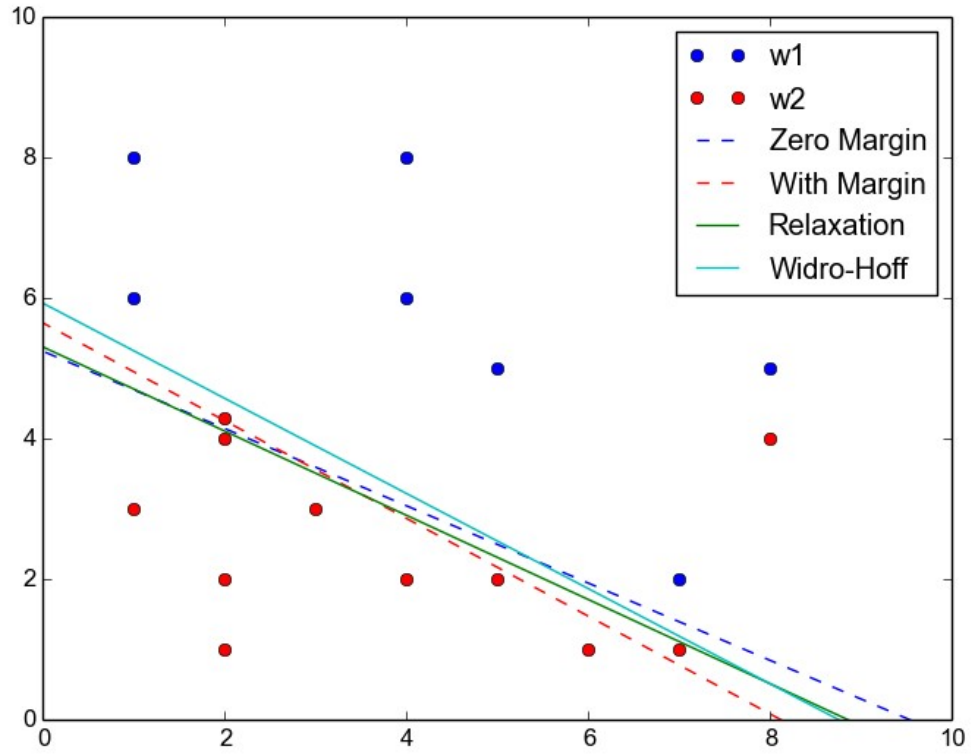
Weight Vector:

Single Sample Perceptron
==========
[[-15.]
 [1.5]
 [2.5]]
Single Sample Perceptron With Margin
==========
[[-23.]
 [2.5]
 [4.]]
Single Sample Perceptron With Relaxation procedure
==========
[[-36.36001475]
 [3.62754536]
 [6.59391461]]
Widro-Hoff LMS
==========
[[233.50636343]
 [-28.03389984]
 [-37.49335998]]

Question: 7
Practical Exercise: B

Adding new samples(non-linearly separable) to the training set

prabhakar@Code$ python Algo_implemented_with_new_data.py



Weight Vector:
Single Sample Perceptron
==========
[[-52.5]
[5.5]
[10.]]
Single Sample Perceptron With Margin
==========
[[-65.]
[8.]
[11.5]]
Single Sample Perceptron With Relaxation procedure
==========
[[-2.45150385e+18]
[2.76562229e+17]
[4.61585832e+17]]
Widro-Hoff LMS
==========
[[232.07787943]
[-26.48489166]
[-39.11830465]]

The nature of the solution found in

Single-sample perceptron
Single-sample perceptron with margin
Relaxation algorithm with margin

As the data is not linearly separable, all the above three algorithms go into an infinite loop. The cause for this behavior is that the above algorithms are trying to find a straight line or an hyperplane that classifies the two classes ω1 and ω2. While trying to do so, always at least one training sample of either or both the classes remains miss-classified. In order to correct it the above algorithm updates the weight vector, which consequently results in the mis-classification of other points. Thus, the above algorithms enter into an infinite loop. The loop is programatically braked after a fixed number of iterations.

Widrow-Hoff
This algorithm converges to a weight vector which produces minimal error with respect to the classification. The convergence time depends on the initial weight vector and also the learning rate that is chosen.

Comparison Table	
Single-sample perceptron	88%
Single-sample perceptron with margin	83%
Relaxation algorithm with margin	88%
Widrow-Hoff	94%

Practical Exercise C:

Relation between the initial weight vector and the convergence time

---

Single-sample perceptron
Single-sample perceptron with margin
Relaxation algorithm with margin
For the above algorithms, if the data is linearly separable the algorithm convergence time is found to be finite irrespective of the initial values of the weight vectors. However, if the data is not linearly separable the convergence time is not finite.

Widrow-Hoff
The algorithm convergence time always changes with respect to the initial values of the weight vectors. If the initial values of the weight vectors in closer to the actual values of the weight vector(that we are supposed to obtain), the algorithm converges faster. On the other hand if the initial values are far away from the actual value of the weight vector, the algorithm takes significantly large time to converge.

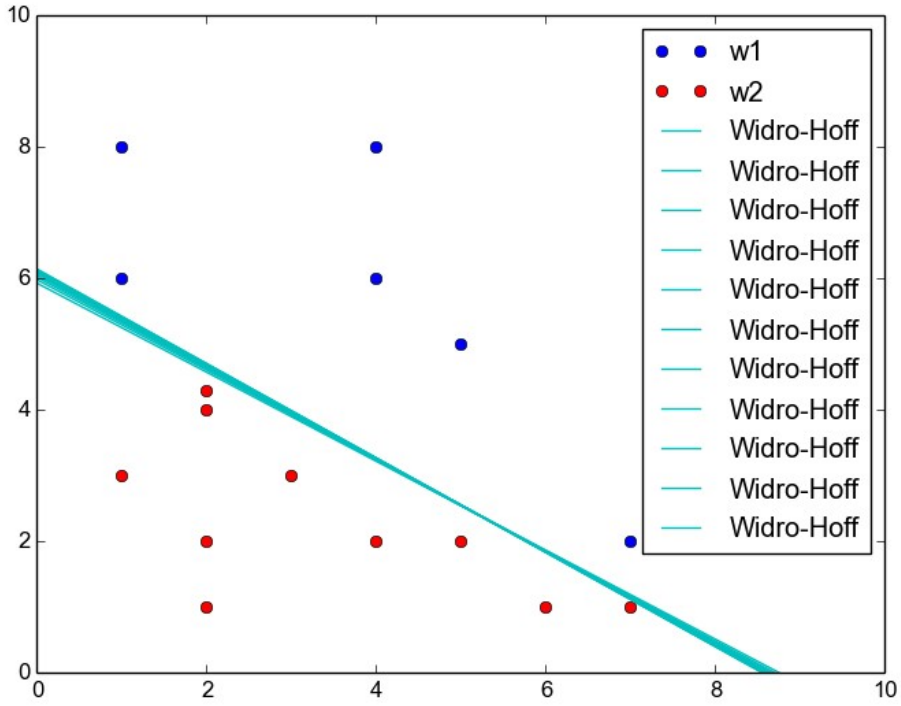
Practical Exercise D:

Effect of adding different margins on the final solution and the convergence-time for algorithms

---

Single-sample perceptron
Single-sample perceptron with margin
Relaxation algorithm with margin
As the margin increases the convergence time is significantly affected for the above algorithms, provided the algorithm always terminates with correct decision boundary.
If we impose a rule stating that the algorithm must terminate after certain number of iterations, the execution time will be still high, however we might not get the accurate or correct decision boundary.

Widrow-Hoff
As the margin changes, the algorithm always terminates with almost same decision boundary without any significant change in the convergence time.



Question 6:

Thus there exists any data-set for which the LMS and perceptron always turns out to be in-line(aligned) or different?

No.
In case of linearly separable data, the solution obtained by both the algorithms will be aligned with each other.
In case of linearly non-separable data, the perceptron algorithm will never converge. So, the depending on the point at which we break the algorithm, the solution could be aligned or different from that of the one obtained from the LMS rule.

Source Code

Algo_Implemented.py

```
"""
        Questions answered:
        ====================
        Implement the following algorithms:
                2. Single-sample perceptron
                3. Single-sample perceptron with margin
                4. Relaxation algorithm with margin
                5. Widrow-Hoff or Least Mean Squared (LMS) Rule

        A. In each case, 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).
"""

#!/usr/bin/python
import sys
from matplotlib import pyplot as ppl
import numpy as np

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)]

w1_x = [ i[0] for i in w1 ]
w1_y = [ i[1] for i in w1 ]
w2_x = [ i[0] for i in w2 ]
w2_y = [ i[1] for i in w2 ]

augmented_Tain_set = [ np.array( [[1], [i[0]], [i[1]]] ) for i in w1 ]
augmented_Tain_set.extend( [ np.array( [[-1], [-i[0]], [-i[1]]] ) for i in w2 ] )

axes = ppl.gca()
axes.set_xlim( [0, max(max(w1_x),max(w2_x)) +1] )
axes.set_ylim( [0, max(max(w1_y),max(w2_y))+1] )

ppl.plot( w1_x, w1_y, "bo", label="w1" )
ppl.plot( w2_x, w2_y, "ro", label="w2" )




def getWeightsForPerceptron( b, perceptronType ):
        if perceptronType=="Widro-Hoff LMS":
                wt_vector = np.array( [[230], [-57.5], [-78.3]] )  # Column Vector
        else:
                wt_vector = np.array( [[0.5], [-0.5], [1.5]] )  # Column Vector
        i=0
```

```
	iteration=0
	while i<len(augmented_Tain_set):
		y = wt_vector.transpose().dot( augmented_Tain_set[i] )
		if perceptronType=="Widro-Hoff LMS":
			etha = 0.01
			wt_vector += etha * ( b -
wt_vector.transpose().dot(augmented_Tain_set[i]) ) * augmented_Tain_set[i];
			i+=1;
		elif y<=b:
			if perceptronType=="Relaxation":
				etha = 2.5
				wt_vector +=  etha * ((float((b - y)) / sum([j[0]*j[0] for j in
augmented_Tain_set[i]])) * augmented_Tain_set[i])
			else:
				etha = 0.5
				wt_vector += etha*augmented_Tain_set[i];
			i=0;
		else:
			i+=1;
		iteration+=1
		if iteration > 100000:
			break;
	return wt_vector;


"""
Single Sample Perceptron
"""
if len(sys.argv)!= 2:
	print "Pass the arguments"
	print "SingleSamplePerceptron"
	print "SingleSamplePerceptronWithMargin"
	print "Relaxation"
	print "LMS"


if sys.argv[1] == "SingleSamplePerceptron" or sys.argv[1] == "all":
	wt_vector = getWeightsForPerceptron( 0, "SingleSamplePerceptron" );
	ppl.plot( [-float(wt_vector[0][0]) / float(wt_vector[1][0]), 0],
				[0, -float(wt_vector[0][0]) / float(wt_vector[2][0])], "b--",
label="Zero Margin")
	print "Single Sample Perceptron"
	print "========================="
	print wt_vector;

if sys.argv[1] == "SingleSamplePerceptronWithMargin" or sys.argv[1] == "all":
	"""
	Single Sample Perceptron With Margin
	"""
```

```
        wt_vector = getWeightsForPerceptron( 1, "SingleSamplePerceptronWithMargin"
);
        ppl.plot( [-float(wt_vector[0][0]) / float(wt_vector[1][0]), 0],
                        [0, -float(wt_vector[0][0]) / float(wt_vector[2][0])], "r--",
label="With Margin")
        print "Single Sample Perceptron With Margin"
        print "======================================="
        print wt_vector;

if sys.argv[1] == "Relaxation" or sys.argv[1] == "all":
        """
        Single Sample Perceptron With Relaxation procedure
        """
        wt_vector = getWeightsForPerceptron( 1, "Relaxation" );
        ppl.plot( [-float(wt_vector[0][0])/float(wt_vector[1][0]), 0],
                        [0, -float(wt_vector[0][0]) / float(wt_vector[2][0])], "g-",
label="Relaxation")
        print "Single Sample Perceptron With Relaxation procedure"
        print
"================================================="
        print wt_vector;

if sys.argv[1] == "LMS" or sys.argv[1] == "all":
        """
        Widro-Hoff LMS
        """
        wt_vector = getWeightsForPerceptron( 1, "Widro-Hoff LMS" );
        ppl.plot( [-float(wt_vector[0][0])/float(wt_vector[1][0]), 0],
                        [0, -float(wt_vector[0][0]) / float(wt_vector[2][0])], "c-",
label="Widro-Hoff LMS")
        print "Widro-Hoff LMS"
        print "=============="
        print wt_vector;

ppl.legend()

if sys.argv[1] != "all":
        ppl.text(1, max(max(w1_y),max(w2_y)), "Weight Vector" )
        ppl.text(1, max(max(w1_y),max(w2_y))-1.1, wt_vector )

# ppl.show();
ppl.savefig('/home/prabhakar/IIIT-H_current/Sem
2/SMAI/Assignments/2/201505618_Assignment2/'+
                                str(sys.argv[1])+'.png', bbox_inches='tight')
```

Algo_implemented_with_new_data.py

```
"""
	Questions answered:
	====================
	B.	Create a test set comprising three more data samples (y i ) for each class and test your
		implementation by computing for each of the test samples the output (class label) predicted
		by the respective algorithm. Create a comparison table listing test set accuracies of each of
		the above algorithms.
"""

#!/usr/bin/python
import sys
from matplotlib import pyplot as ppl
import numpy as np

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), (8, 4)]

# Adding new data elements
w1.extend( [(4, 6), (1, 8), (5, 5)] )
w2.extend( [(2, 2), (4, 2), (6, 1), (2, 4.3)] )

w1_x = [ i[0] for i in w1 ]
w1_y = [ i[1] for i in w1 ]
w2_x = [ i[0] for i in w2 ]
w2_y = [ i[1] for i in w2 ]

augmented_Tain_set = [ np.array( [[1], [i[0]], [i[1]]] ) for i in w1 ]
augmented_Tain_set.extend( [ np.array( [[-1], [-i[0]], [-i[1]]] ) for i in w2 ] )

axes = ppl.gca()
axes.set_xlim( [0, max(max(w1_x),max(w2_x)) +1] )
axes.set_ylim( [0, max(max(w1_y),max(w2_y))+1] )

ppl.plot( w1_x, w1_y, "bo", label="w1" )
ppl.plot( w2_x, w2_y, "ro", label="w2" )



def getWeightsForPerceptron( b, perceptronType ):
	if perceptronType=="Widro-Hoff LMS":
		wt_vector = np.array( [[230], [-57.5], [-78.3]] )  # Column Vector
	else:
		wt_vector = np.array( [[0.5], [-0.5], [1.5]] )  # Column Vector
	i=0
	iteration=0
```

```
        while i<len(augmented_Tain_set):
                y = wt_vector.transpose().dot( augmented_Tain_set[i] )
                if perceptronType=="Widro-Hoff LMS":
                        etha = 0.01
                        wt_vector += etha * ( b -
wt_vector.transpose().dot(augmented_Tain_set[i]) ) * augmented_Tain_set[i];
                        i+=1;
                elif y<=b:
                        if perceptronType=="Relaxation":
                                etha = 2.5
                                wt_vector +=  etha * ((float((b - y)) / sum([j[0]*j[0] for j in
augmented_Tain_set[i]])) * augmented_Tain_set[i])
                        else:
                                etha = 0.5
                                wt_vector += etha*augmented_Tain_set[i];
                        i=0;
                else:
                        i+=1;
                iteration+=1
                if iteration > 100000:
                        break;
        return wt_vector;


"""
Single Sample Perceptron
"""
wt_vector = getWeightsForPerceptron( 0, "singleSamplePerceptron" );
ppl.plot( [-float(wt_vector[0][0]) / float(wt_vector[1][0]), 0],
                  [0, -float(wt_vector[0][0]) / float(wt_vector[2][0])], "b--", label="Zero
Margin")
print "Single Sample Perceptron"
print "========================="
print wt_vector;


"""
Single Sample Perceptron With Margin
"""
wt_vector = getWeightsForPerceptron( 1, "SingleSamplePerceptronWithMargin" );
ppl.plot( [-float(wt_vector[0][0]) / float(wt_vector[1][0]), 0],
                  [0, -float(wt_vector[0][0]) / float(wt_vector[2][0])], "r--", label="With
Margin")
print "Single Sample Perceptron With Margin"
print "===================================="
print wt_vector;


"""
Single Sample Perceptron With Relaxation procedure
```

```
"""
wt_vector = getWeightsForPerceptron( 1, "Relaxation" );
ppl.plot( [-float(wt_vector[0][0])/float(wt_vector[1][0]), 0],
                    [0, -float(wt_vector[0][0]) / float(wt_vector[2][0])], "g-",
label="Relaxation")
print "Single Sample Perceptron With Relaxation procedure"
print
"================================================="
print wt_vector;


"""
Widro-Hoff
"""
wt_vector = getWeightsForPerceptron( 1, "Widro-Hoff LMS" );
ppl.plot( [-float(wt_vector[0][0])/float(wt_vector[1][0]), 0],
                    [0, -float(wt_vector[0][0]) / float(wt_vector[2][0])], "c-", label="Widro-
Hoff")
print "Widro-Hoff LMS"
print "=============="
print wt_vector;


ppl.legend()

ppl.savefig('/home/prabhakar/IIIT-H_current/Sem
2/SMAI/Assignments/2/201505618_Assignment2/'+
                                    str(sys.argv[0])+'.png', bbox_inches='tight')
# ppl.show();
```