

## CMPE 257 (ML) ASSIGNMENT 1

RISHIKA MACHINA

SID: 012525227

rishika.machina@sjsu.edu

### EXERCISE PROBLEMS

**1.3** The weight update rule, which has the nice interpretation that it moves in the direction of classifying  $x(t)$  correctly.

(a) Show that  $y(t)w^T(t)x(t) < 0$ . [Hint:  $x(t)$  is misclassified by  $w(t)$ .]

(b) Show that  $y(t)w^T(t+1)x(t) > y(t)w^T(t)x(t)$ . [Hint: Use (1.3).]

(c) As far as classifying  $x(t)$  is concerned, argue that the move from  $w(t)$  to  $w(t+1)$  is a move 'in the right direction'.

### ANS

(a) Show that  $y(t)w^T(t)x(t) < 0$ . [Hint:  $x(t)$  is misclassified by  $w(t)$ .]

According to the perceptron algorithm,  $Y(t) = \text{sign}(w^T(t)x(t))$

Where,  $\text{sign}(s) = +1$  for  $s \geq 0$  and  $-1$  for  $s < 0$

If  $x(t)$  is misclassified,  $y(t)$  is not equal to  $\text{sign}(w^T(t)x(t))$ , which means both are of opposite signs since two possible outputs are  $+1$  and  $-1$ .

(i.e)  $y(t) = 1$  if  $\text{sign}(w^T(t)x(t)) = -1$  and  $y(t) = -1$  if  $\text{sign}(w^T(t)x(t)) = 1$

Therefore, for a misclassified example,  $y(t)w^T(t)x(t) < 0$ .

(b) Show that  $y(t)w^T(t+1)x(t) > y(t)w^T(t)x(t)$ . [Hint: Use (1.3).]

Weight is updated based on the given rule,

$$w(t+1) = w(t) + y(t)x(t)$$

Using the above equation in  $y(t)w^T(t+1)x(t) = y(t)x(t)[w^T(t) + y(t)x^T(t)] = y(t)x(t)w^T(t) + y^2(t)x^T(t)x(t)$ , which is evidently greater than  $y(t)w^T(t)x(t)$ .

(c) As far as classifying  $x(t)$  is concerned, argue that the move from  $w(t)$  to  $w(t + 1)$  is a move 'in the right direction'.

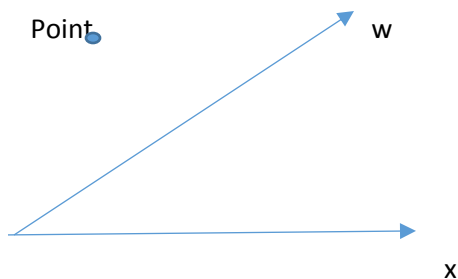
Consider a misclassified example, with input  $x(t)$  and initial weight  $w(t)$ . They are said to be misclassified only if  $y(t) = -1$  where it has to be  $+1$  or vice versa.

Two cases are possible

**First case:**

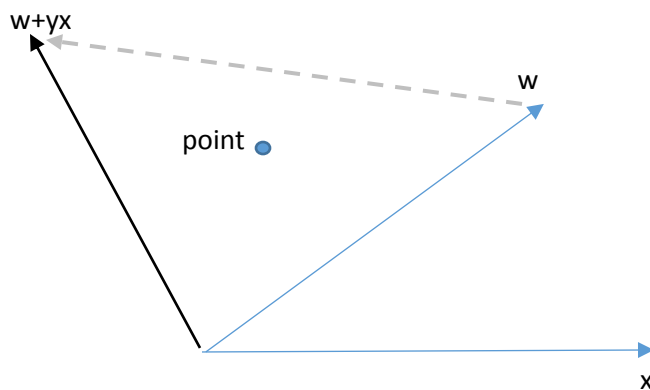
Let ' $P$ ' be the angle between  $w$  and  $x$

For  $0 < P < (\pi/2)$  and  $(3\pi/2) < P < 2\pi$



Here  $y$  should be  $+1$  but as the point is misclassified.  $Y$  is  $-1$ .

In order to change  $y$  as  $+1$ , If we attempt to add  $yx$  to the  $w$  available. The following change will occur.

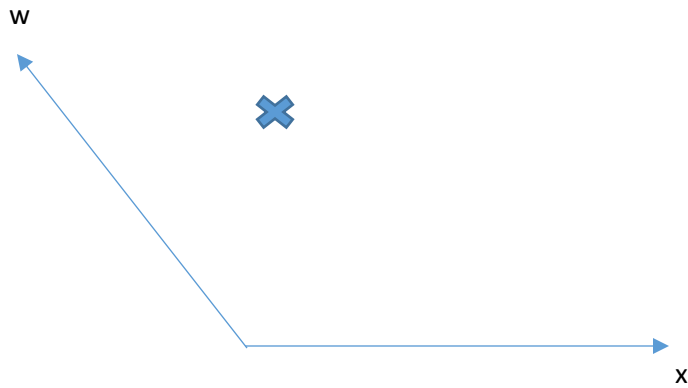


After changing the weight to  $w+yx$ ,  $y$  for the point will be  $+1$ .

Hence for this case, changing the weight according the rule is correct.

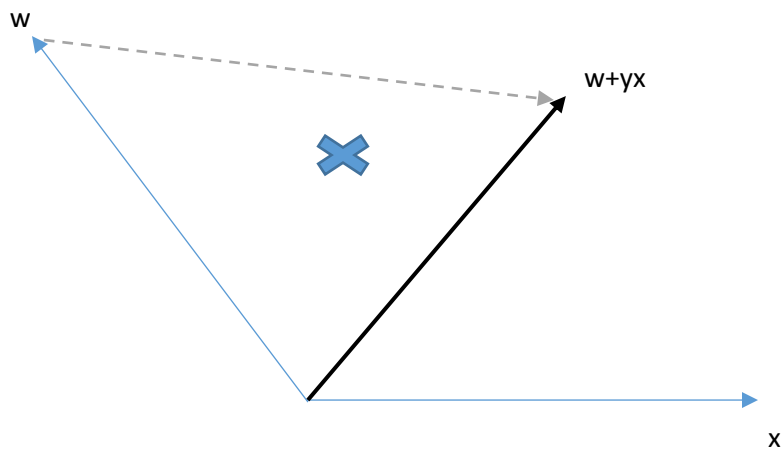
**Second case:**

$$(\pi/2) < P < (3\pi/2)$$



For this case  $y$  should be -1 but it is +1, since it is a misclassified example.

After changing weight as  $w+yx$ , the figure will change as shown.



After changing the weight to  $w+yx$ ,  $y$  for the point will be -1.

Hence for this case, changing the weight according the rule is correct.

**Finally, changing  $w(t)$  to  $w(t+1)$  will move in the right direction.**

**1.6** For each of the following tasks, identify which type of learning is involved (supervised, reinforcement, or unsupervised) and the training data to be used. If a task can fit more than one type, explain how and describe the training data for each type.

(a) Recommending a book to a user in an online bookstore

(b) Playing tic tac toe

(c) Categorizing movies into different types

(d) Learning to play music

(e) Credit limit: Deciding the maximum allowed debt for each bank customer

**ANS**

(a) Recommending a book to a user in an online bookstore is purely '**supervised learning**' since the model will be based on the previous customers and choices made by them.

(b) Playing tic tac toe could be '**Reinforced or Unsupervised learning**' since the machine should either be fed with the game rules or it should learn by its own.

(c) Categorizing movies into different types is purely '**Supervised learning**' since it is based on past movies and their categorization.

(d) Learning to play music can be '**Reinforced or Unsupervised learning**' since the machine should be fed with basics of music or should learn on its own.

(e) Credit limit is purely '**Supervised learning**' as the required factors change from bank to bank and the machine either approves or declines based on the previous data.

**1.8** If  $\mu = 0.9$ , what is the probability that a sample of 10 marbles will have  $v \leq 0.1$ ?

[Hints: 1. Use binomial distribution. 2. The answer is a very small number.]

**ANS**

We should find the probability of  $v \leq 0.1$ , which means the number of red marbles in the sample should be either 0 or 1. So we should find the sum of probabilities for  $v=0$  and  $v=1$ .

Binomial distribution,  $P[X=r] = \binom{n}{r} * p^r * q^{n-r}$

Let 'X' be the number of red marbles picked whereas 'v' is the probability of red marbles in the sample

$$\begin{aligned}
 P[v \leq 0.1] &= P[X=0] + P[X=1] = \binom{10}{0} * (0.9)^0 * (0.1)^{10} + \binom{10}{1} * (0.9)^1 * (0.1)^9 \\
 &= 1 * 1 * 0.1^{10} + 10 * 0.9 * 0.1^9 \\
 &= 0.1^{10} + 10 * 0.9 * 0.1^9 \\
 &= 0.1^{10} + 90 * 0.1^{10} \\
 &= 91 * 10^{-10}
 \end{aligned}$$

**1.9** If  $\mu = 0.9$ , use the Hoeffding Inequality to bound the probability that a sample of 10 marbles will have  $v \leq 0.1$  and compare the answer to the previous exercise.

**ANS**

We know that Hoeffding inequality is  $P[|v-u| > E] \leq 2 * \exp(-2E^2N)$

We know that if  $|A| < 1$  implies  $-1 < A < +1$ . Using this for the left part of the Hoeffding inequality

$$|v-u| > E \longrightarrow -E < v-u < E \longrightarrow (-E+u) < v < (E+u) \longrightarrow (-E+0.9) < v < (E+0.9)$$

To find  $P(v \leq 0.1)$ ,  $E+0.9=0.1 \longrightarrow E=0.1-0.9=-0.8$

$$\text{Finally, } P[v \leq 0.1] \leq 2 * \exp(-2 * (-0.8)^2 * 10) \leq 2 * \exp(-12.8) \leq 5.52 * 10^{-6}$$

Therefore, using Hoeffding inequality,  $P[v \leq 0.1] \leq 5.52 * 10^{-6}$

The probability obtained using **binomial** distribution is very **less than** that obtained using **Hoeffding** inequality.

## PROBLEMS

**1.2** Consider the perceptron in two dimensions:  $h(x) = \text{sign}(w^T x)$  where  $w = [w_0, w_1, w_2]^T$  and  $x = [1, x_1, x_2]^T$ . Technically,  $x$  has three coordinates, but we call this perceptron two-dimensional because the first coordinate is fixed at 1.

(a) Show that the regions on the plane where  $h(x) = +1$  and  $h(x) = -1$  are separated by a line. If we express this line by the equation  $x_2 = ax_1 + b$ , what are the slope  $a$  and intercept  $b$  in terms of  $w_0, w_1, w_2$ ?

(b) Draw a picture for the cases  $w = [1, 2, 3]^T$  and  $w = -[1, 2, 3]^T$ .

In more than two dimensions, the  $+1$  and  $-1$  regions are separated by a hyperplane, the generalization of a line.

## ANS

a)  $H(x) = \text{sign}(w^T x)$

$$W^T x = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = w_0 + (w_1 * x_1) + (w_2 * x_2)$$

We know that  $\text{sign}(x) = -1$  for negative  $x$  and is  $+1$  for positive  $x$ , which means  $\text{sign}(w^T x) = 0$  will differentiate both the regions.

$$w_0 + (w_1 * x_1) + (w_2 * x_2) = 0 \longrightarrow x_2 = (-w_1/w_2)x_1 + (-w_0/w_2)$$

Comparing it with  $x_2 = a.x_1 + b$ , **Slope(a) =  $-w_1/w_2$**  and **Intercept(b) =  $-w_0/w_2$**

b) When  $w = [1 \ 2 \ 3]^T$  line will be  $1 + 2*x_1 + 3*x_2 = 0$

plots for both  $w$  set is the same but the region only differ

c) When  $w = [-1 \ -2 \ -3]^T$  line will be  $-1 - 2*x_1 - 3*x_2 = 0 \rightarrow 1 + 2*x_1 + 3*x_2 = 0$

**1.4** In Exercise 1.4, we use an artificial data set to study the perceptron learning algorithm. This problem leads you to explore the algorithm further with data sets of different sizes and dimensions.

(a) Generate a linearly separable data set of size 20 as indicated in Exercise 1.4. Plot the examples  $\{(x_n, Y_n)\}$  as well as the target function  $f$  on a plane. Be sure to mark the examples from different classes differently, and add labels to the axes of the plot.

(b) Run the perceptron learning algorithm on the data set above. Report the number of updates that the algorithm takes before converging. Plot the examples  $\{(x_n, Y_n)\}$ , the target function  $f$ , and the final hypothesis  $g$  in the same figure. Comment on whether  $f$  is close to  $g$ .

(c) Repeat everything in (b) with another randomly generated data set of size 20. Compare your results with (b).

(d) Repeat everything in (b) with another randomly generated data set of size 100. Compare your results with (b).

(e) Repeat everything in (b) with another randomly generated data set of size 1000. Compare your results with (b).

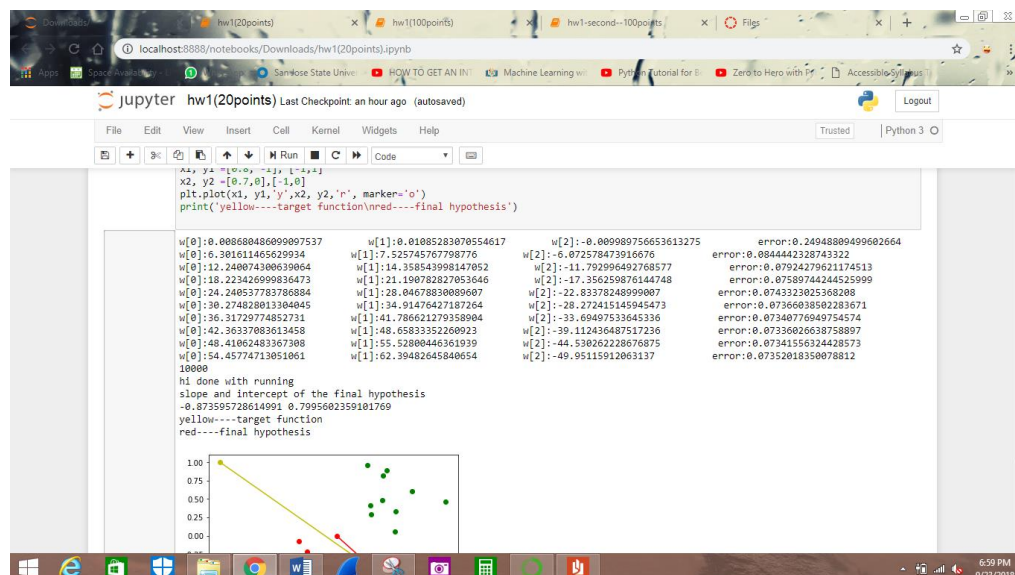
## ANS

For the given data set, two major differences were observed,

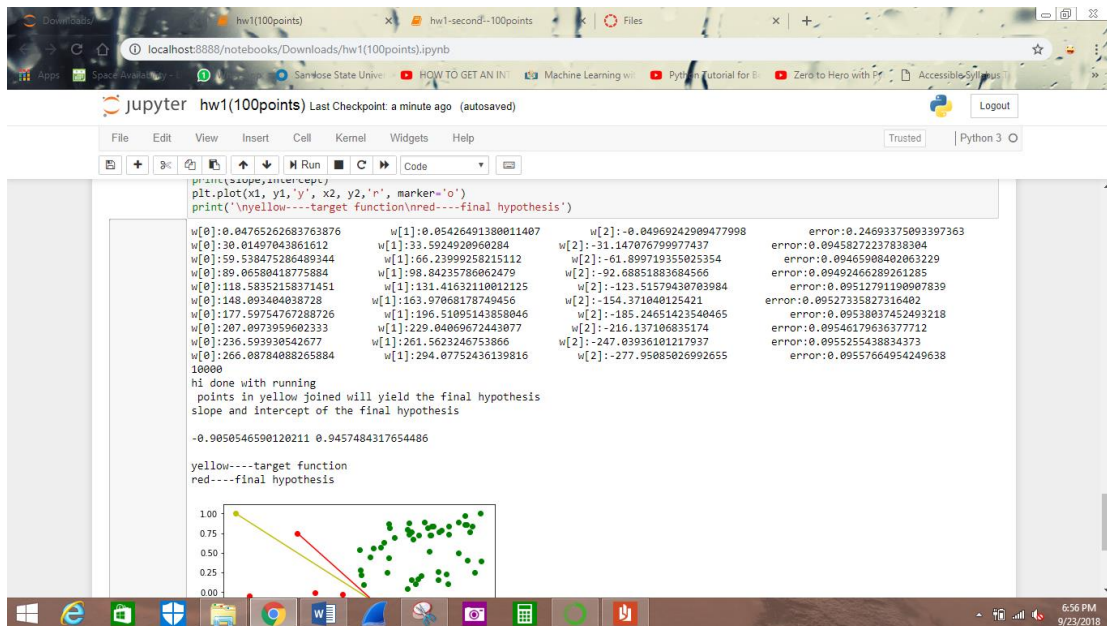
- 1) slope of the final hypothesis is different even if the data is in first and third quadrant
- 2) running time is different

‘.pynb file’ for 3 cases is submitted.

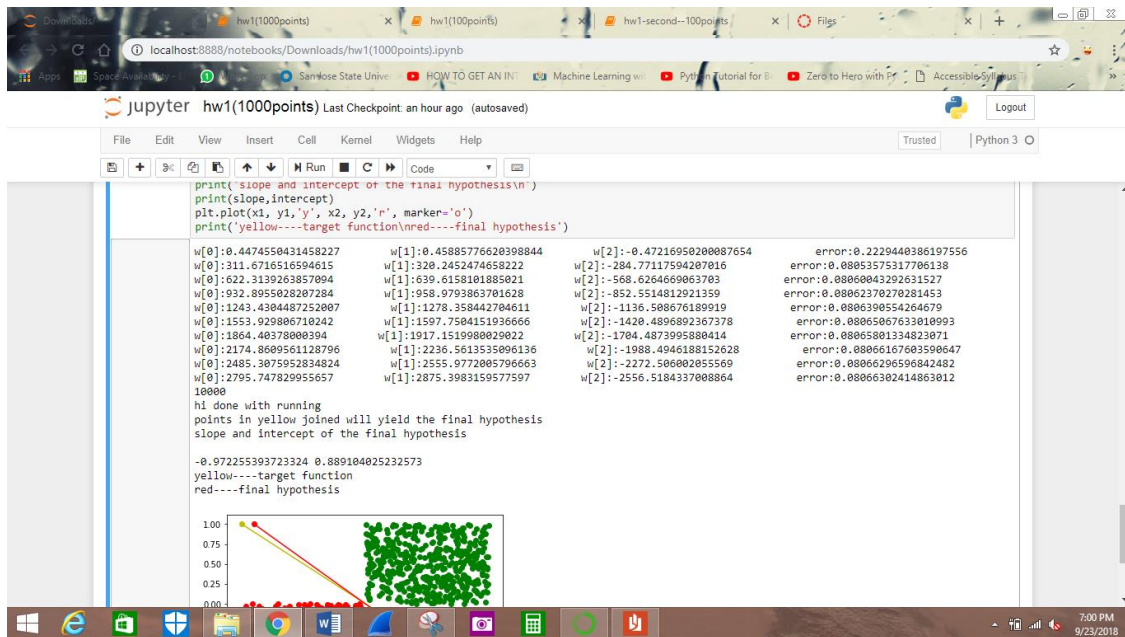
**20 points output:**



## 100 points output:



## 1000 points output:





**1.5** The perceptron learning algorithm works like this: In each iteration  $t$ , pick a random  $(x(t), y(t))$  and compute the 'signal'  $s(t) = w^T(t)x(t)$ . If  $y(t) \cdot s(t) \leq 0$ , update  $w$  by

$$w(t+1) = w(t) + y(t) \cdot x(t) ;$$

One may argue that this algorithm does not take the 'closeness' between  $s(t)$  and  $y(t)$  into consideration. Let's look at another perceptron learning algorithm: In each iteration, pick a random  $(x(t), y(t))$  and compute  $s(t)$ . If  $y(t) \cdot s(t) \leq 1$ , update  $w$  by

$$w(t+1) = w(t) + n \cdot (y(t) \cdot s(t)) \cdot x(t) ,$$

where  $n$  is a constant. That is, if  $s(t)$  agrees with  $y(t)$  well (their product is  $> 1$ ), the algorithm does nothing. On the other hand, if  $s(t)$  is further from  $y(t)$ , the algorithm changes  $w(t)$  more. In this problem, you are asked to implement this algorithm and study its performance.

(a) Generate a training data set of size 100 similar to that used in Exercise 1.4. Generate a test data set of size 10,000 from the same process. To get  $g$ , run the algorithm above with  $n = 100$  on the training data set, until a maximum of 1,000 updates has been reached. Plot the training data set, the target function  $f$ , and the final hypothesis  $g$  on the same figure. Report the error on the test set.

(b) Use the data set in (a) and redo everything with  $n = 1$ .

(c) Use the data set in (a) and redo everything with  $n = 0.01$ .

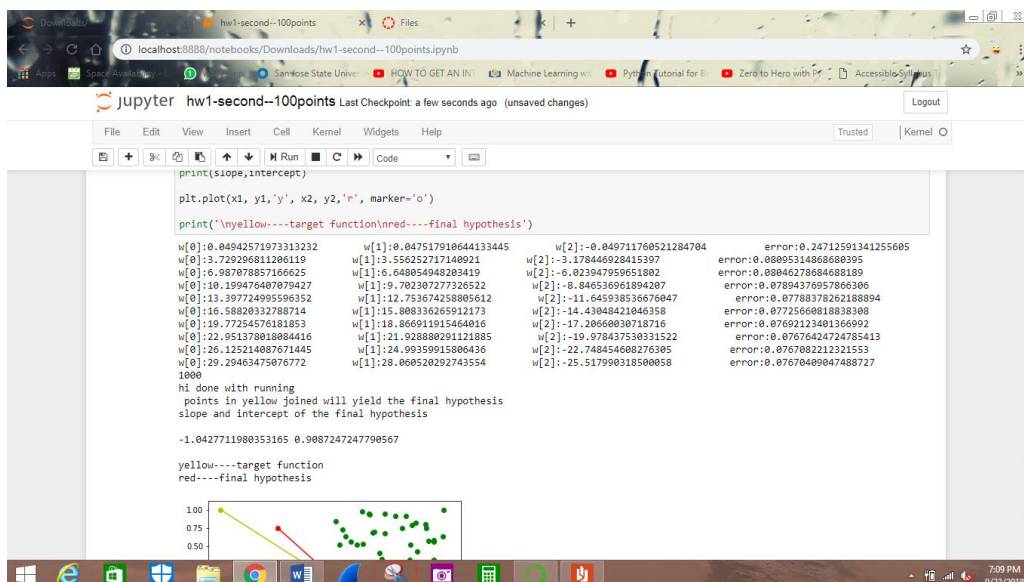
(d) Use the data set in (a) and redo everything with  $n = 0.0001$ .

(e) Compare the results that you get from (a) to (d).

The algorithm above is a variant of the so called Adaline (Adaptive Linear Neuron) algorithm for perceptron learning.

**ANS** Error is decreased a bit when compared to 1.4 problem

For 100 points— $n=100$



n= 1

```
slope=(-p.w[0]/p.w[1])
intercept=-p.w[2]/p.w[1]

print('slope and intercept of the final hypothesis\n')
print(slope,intercept)

plt.plot(x1, y1,'y', x2, y2,'r', marker='o')

print('\nyellow----target function\nred----final hypothesis')

w[0]:0.04982069757172051 w[1]:0.05030107008944584 w[2]:-0.04969601614449259 error:0.2469694020633668
w[0]:3.7062922941909093 w[1]:3.7195451509434645 w[2]:-3.2704080149709975 error:0.08892229757862385
w[0]:6.967490465775113 w[1]:6.918032486550281 w[2]:-6.25332837235746 error:0.0882919711189488
w[0]:10.21037957973746 w[1]:10.05949561914805 w[2]:-9.20632361737777 error:0.006186158619828
w[0]:13.459727021904573 w[1]:13.18432531083402 w[2]:-12.12756165342367 error:0.0046005014799684
w[0]:16.717478395172943 w[1]:16.30284700853001 w[2]:-15.026184920292987 error:0.00352287958596104
w[0]:19.98209892584543 w[1]:19.418675448078638 w[2]:-17.909510480812894 error:0.00280659395841117
w[0]:23.25172824193302 w[1]:22.533089326313902 w[2]:-20.78273646542094 error:0.00233997742454903
w[0]:26.524789105835964 w[1]:25.646523684076914 w[2]:-23.64943135492066 error:0.00204538914825367
w[0]:29.80007955926573 w[1]:28.759104961944164 w[2]:-26.512091104709015 error:0.00186896378539718
1000
hi done with running
points in yellow joined will yield the final hypothesis
slope and intercept of the final hypothesis
-1.037820877618989 0.921610052505904

yellow----target function
red----final hypothesis
```

n=0.01

```
x2, y2 = [0.75, -0.5], [-1, 0.75]
slope=(-p.w[0]/p.w[1])
intercept=-p.w[2]/p.w[1]

print('slope and intercept of the final hypothesis\n')
print(slope,intercept)

plt.plot(x1, y1,'y', x2, y2,'r', marker='o')

print('\nyellow----target function\nred----final hypothesis')

w[0]:0.048310672687030994 w[1]:0.05125564406649871 w[2]:-0.04969202812092265 error:0.24692976587705667
w[0]:3.608208535185347 w[1]:3.8367872091584023 w[2]:-3.214241023576398 error:0.08478495080005427
w[0]:6.768394682697087 w[1]:7.18525646673724 w[2]:-6.13461645706139 error:0.08516509507422303
w[0]:9.88453490312713 w[1]:10.488757596453889 w[2]:-9.044968939966397 error:0.08427155631866012
w[0]:12.985926869286788 w[1]:13.779150392409559 w[2]:-11.94202436731181 error:0.08363725777478546
w[0]:16.08042431182967 w[1]:17.065622612862867 w[2]:-14.82998274496217 error:0.08319478103428626
w[0]:19.171371781018717 w[1]:20.35155643514706 w[2]:-17.71116726001271 error:0.08284364025289775
w[0]:22.260485585286713 w[1]:23.638537719809193 w[2]:-20.586714320807424 error:0.08254137603928795
w[0]:25.34867553730833 w[1]:26.927318432209226 w[2]:-23.457322698106886 error:0.08227470327619602
w[0]:28.436390321467425 w[1]:30.218209463578372 w[2]:-26.323569786457323 error:0.0820404137020271
1000
hi done with running
points in yellow joined will yield the final hypothesis
slope and intercept of the final hypothesis
-0.9406958449074239 0.8709331485500501

yellow----target function
red----final hypothesis
```

n=0.0001

```

slope = (p.w[0]/p.w[1])
intercept = p.w[2]/p.w[1]

print('slope and intercept of the final hypothesis\n')
print(slope, intercept)

plt.plot(x1, y1, 'y', x2, y2, 'r', marker='o')

print('\nyellow---target function\nred---final hypothesis')

w[0]: 0.84807246444912078 w[1]: 0.845007165707000734 w[2]: -0.404726774097190186 error: 0.2472752147643028
w[3]: 3.6704182289965741 w[4]: 3.507270855333416 w[5]: -3.1525515549258056 error: 0.0774895970277934
w[6]: 6.834297736042195 w[7]: 6.627121154000118 w[8]: -5.918099055778664 error: 0.07520424181482707
w[9]: 9.950818018535002 w[10]: 9.728131227029067 w[11]: -8.641009927339072 error: 0.0717742629784006
w[12]: 13.055977951333459 w[13]: 12.801859172398 w[14]: -11.312879926393009 error: 0.0718624008067909
w[15]: 16.15850152045137 w[16]: 15.932676849566718 w[17]: -14.005599918382025 error: 0.07108482279062381
w[18]: 19.259173286200235 w[19]: 18.939133841160603 w[20]: -16.067205976318385 error: 0.07064679488272743
w[21]: 22.358032341973757 w[22]: 22.147282939702173 w[23]: -19.327738046093456 error: 0.07041924051306083
w[24]: 25.45800265958612 w[25]: 25.2565488889795 w[26]: -21.975285139984216 error: 0.0701793954512174
w[27]: 28.55812905604195 w[28]: 28.36643593738874 w[29]: -24.82668372081587 error: 0.07028951956945002
1000
hi done with running
points in yellow joined will yield the final hypothesis
slope and intercept of the final hypothesis
-1.005315320889058 0.86626585639227
yellow---target function
red---final hypothesis

```

**1.11** The matrix which tabulates the cost of various errors for the CIA and Supermarket applications in Example 1.1 is called a risk or loss matrix.

For the two risk matrices in Example 1.1, explicitly write down the in sample error  $E_{in}$  that one should minimize to obtain  $g$ . This in-sample error should weight the different types of errors based on the risk matrix.

[Hint: Consider  $Y_n = +1$  and  $Y_n = -1$  separately.]

**ANS**

We know that  $E_{in}$  is 'In sample error'

for Supermarket

$$E_{in}(h) = \frac{1}{N} \sum_{n=1}^N [h(x_n) \mp f(x_n)]$$

$$= \frac{1}{N} \left[ \sum_{y_n=1} e(h(x_n), 1) + \sum_{y_n=-1} e(h(x_n), -1) \right]$$

$$= \frac{1}{N} \left[ \sum_{y_n=1} 10(h(x_n) \mp y_n) + \sum_{y_n=-1} (h(x_n) \mp y_n) \right]$$

Similarly, for CIA.

$$E_{in}(h) = \frac{1}{N} \left[ \sum_{y_n=1} (h(x_n) \mp y_n) + 1000 \sum_{y_n=-1} (h(x_n) \mp y_n) \right]$$