

# Machine Learning (ENME808E): Homework 2

Koyal Bhartia(116350990)

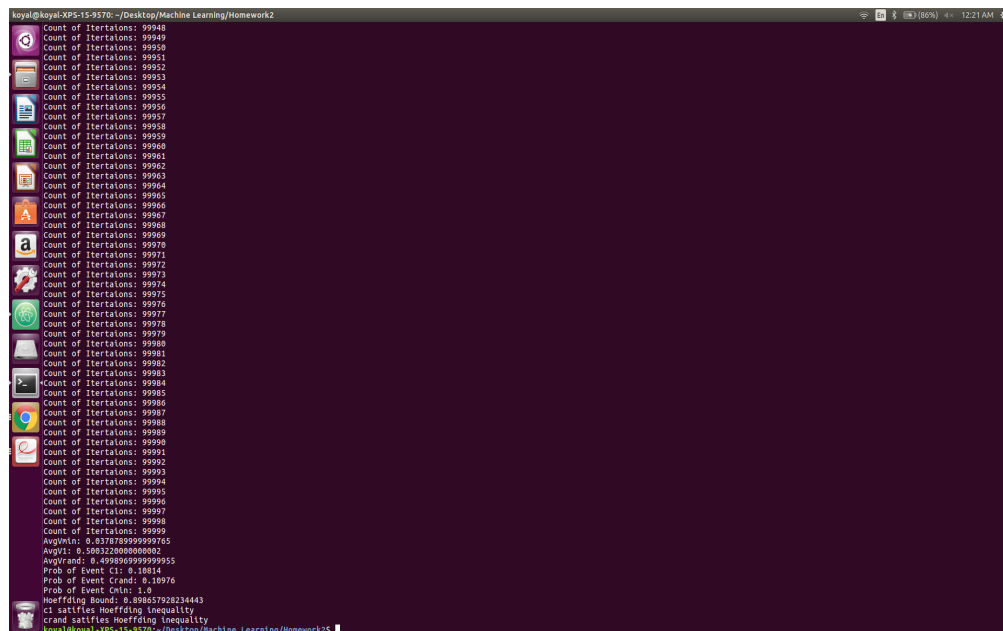
March 8, 2019

## 1 Hoeffding Inequality - Question 1 and 2

Please find the code attached for Hoeffding Inequality in Section 6 - Hoeffding Inequality using Virtual Coins

Run a computer simulation for flipping 1,000 virtual fair coins. Flip each coin independently 10 times. Focus on 3 coins as follows:  $c_1$  is the first coin flipped,  $c_{rand}$  is a coin chosen randomly from the 1,000, and  $c_{min}$  is the coin which had the minimum frequency of heads (pick the earlier one in case of a tie). Let  $v_1$ ,  $v_{rand}$ , and  $v_{min}$  be the fraction of heads obtained for the 3 respective coins out of the 10 tosses. Run the experiment 100,000 times in order to get a full distribution of  $v_1$ ,  $v_{rand}$ , and  $v_{min}$  (note that  $c_{rand}$  and  $c_{min}$  will change from run to run).

Following is the terminal screenshot obtained after running the code written in Python for questions 1 and 2.



```
koyal@koyal-XPS-15-9570: ~/Desktop/Machine Learning/Homework2
Count of Iterations: 99948
Count of Iterations: 99949
Count of Iterations: 99950
Count of Iterations: 99951
Count of Iterations: 99952
Count of Iterations: 99953
Count of Iterations: 99954
Count of Iterations: 99955
Count of Iterations: 99956
Count of Iterations: 99957
Count of Iterations: 99958
Count of Iterations: 99959
Count of Iterations: 99960
Count of Iterations: 99961
Count of Iterations: 99962
Count of Iterations: 99963
Count of Iterations: 99964
Count of Iterations: 99965
Count of Iterations: 99966
Count of Iterations: 99967
Count of Iterations: 99968
Count of Iterations: 99969
Count of Iterations: 99970
Count of Iterations: 99971
Count of Iterations: 99972
Count of Iterations: 99973
Count of Iterations: 99974
Count of Iterations: 99975
Count of Iterations: 99976
Count of Iterations: 99977
Count of Iterations: 99978
Count of Iterations: 99979
Count of Iterations: 99980
Count of Iterations: 99981
Count of Iterations: 99982
Count of Iterations: 99983
Count of Iterations: 99984
Count of Iterations: 99985
Count of Iterations: 99986
Count of Iterations: 99987
Count of Iterations: 99988
Count of Iterations: 99989
Count of Iterations: 99990
Count of Iterations: 99991
Count of Iterations: 99992
Count of Iterations: 99993
Count of Iterations: 99994
Count of Iterations: 99995
Count of Iterations: 99996
Count of Iterations: 99997
Count of Iterations: 99998
Count of Iterations: 99999
Avgv1n: 0.5378789999999765
Avgvrand: 0.528323080000002
Avgvmin: 0.4998969999999955
Prob of Event C1: 0.18814
Prob of Event Cmin: 1.0
Hoeffding Bound: 0.8885782824443
c1 satisfies Hoeffding Inequality
cmin satisfies Hoeffding Inequality
koyal@koyal-XPS-15-9570: ~/Desktop/Machine Learning/Homework2
```

Figure 1: Single bin Hoeffding Inequality on 1000 virtual coins

## 1.1 Question 1

The average value of  $\nu_{min}$  is closest to: [a] 0 [b] 0.01 [c] 0.1 [d] 0.5 [e] 0.67

**ANSWER :[B]**

The screenshot shows the full distribution of  $\nu$  obtained for all the 3 events:  $C_1$ ,  $C_{rand}$  and  $C_{min}$ . This is done by summing the value of the fraction of the heads among the number of the flips of each coin for each of the iterations and then averaging over the total no of iterations to get the Average value of  $\nu$ . Clearly the value of  $\nu_{min}$  obtained is: **0.037**; which among the given options is closest to 0.01. Hence the answer is 0.01.

## 1.2 Question 2

Which coin(s) has a distribution of  $\nu$  that satisfies the (single-bin) Hoeffding Inequality? [a]  $c_1$  only [b]  $c_{rand}$  only [c]  $c_{min}$  only [d]  $c_1$  and  $c_{rand}$  [e]  $c_{min}$  and  $c_{rand}$

**ANSWER:[D]**

As given, the 3 events considered are:  $C_1$ ,  $C_{rand}$  and  $C_{min}$  (coin with min freq of heads). The following Hoeffding inequality is used to determine which distribution of  $\nu$  satisfies the same:

$$P[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N} \quad (1)$$

The values of  $\nu_{max}, \nu_1, \nu_{rand}$  is already calculated for 1000 iterations in the previous problem. The  $\nu$  obtained in each iteration is used to calculate the corresponding probability which is then averaged over 1000, to check if the event satisfies the Hoeffding inequality.

As the coins are virtual and unbiased, following assumptions have been made:  $\mu=0.5$   $\epsilon=0.2$

Thus the value of  $2e^{-2\epsilon^2 N}$  obtained is 0.89

The corresponding probability for each event is calculated using:  $P[|\nu - \mu| > \epsilon]$  As shown in the screenshot, the probability values obtained are as follows:

$$P_1 = P[|\mu - \nu_1| > \epsilon] = 0.108$$

$$P_{rand} = P[|\mu - \nu_{rand}| > \epsilon] = 0.109$$

$$P_{min} = P[|\mu - \nu_{min}| > \epsilon] = 1.0$$

Thus we see that,  $P_1$  and  $P_{rand}$  is lesser than the calculated Hoeffding bound i.e 0.89

Hence the answer is:  $C_1$  and  $C_{rand}$  will follow Hoeffding Inequality.

## 2 Error and Noise - Question 3 and 4

Consider the bin model for a hypothesis  $h$  that makes an error with probability  $\mu$  in approximating a deterministic target function  $f$  (both  $h$  and  $f$  are binary functions). If we use the same  $h$  to approximate a noisy version of  $f$  given by:  $P(y/x) = \lambda$  for  $y = f(x)$  and  $1 - \lambda$  for  $y \neq f(x)$

## 2.1 Question 3

What is the probability of error that  $h$  makes in approximating  $y$ ? Hint: Two wrongs can make a right! [a]  $\mu$  [b]  $\lambda$  [c]  $1-\mu$  [d]  $(1-\lambda)*\mu+\lambda*(1-\mu)$  [e]  $(1-\lambda)*(1-\mu)+\lambda*\mu$

**ANSWER[E]**

It is given that the probability with which  $h$  makes an error is  $\mu$

Hence the probability with which  $h$  is true in  $1 - \mu$

It is also given that:  $P(y|x)$  is  $1 - \lambda$  for  $y \neq f(x)$  and  $P(y=x)$  is  $\lambda$  for  $y=f(x)$

Calculating the total error probability can be split into the sum of 2 cases:

**Case 1:** Probability of  $h$  making error(false answer due to hypotheses) is  $\mu$

and  $P(y|x)$  getting right(point not miss classified due to noise) is  $\lambda$

Thus, Error probability1 =  $\mu \lambda$

**Case 2:** Probability of  $h$  making an error(false answer due to hypotheses) is  $(1 - \mu)$  and  $P(y|x)$  getting wrong(point is miss classified due to noise) is  $(1 - \lambda)$

Thus, Error probability2 =  $(1 - \mu) (1 - \lambda)$

Hence, **Total Error probability** = **Case1+Case2** =  $\mu * \lambda + (1 - \mu) * (1 - \lambda)$

## 2.2 Question 4

At what value of  $\lambda$  will the performance of  $h$  be independent of  $\mu$ ? [a] 0 [b] 0.5 [c]  $1/\sqrt{2}$  [d] 1 [e] No values of  $\lambda$

**ANSWER :[B]**

Using the probability of error that  $h$  makes obtained above:

$$\mu * \lambda + (1 - \mu) * (1 - \lambda)$$

For  $h$  to be independent of  $\mu$ , we need a condition such that the value is same for  $\mu$  and  $1 - \mu$ .

Hence equating:  $\lambda = (1 - \lambda)$ , we get  $\lambda = 0.5$

Substitute this value of  $\lambda$  back in the equation of performance of  $h$  we get:

Total Error probability

$$= \mu * 0.5 + (1 - \mu) * 0.5 = \mu * 0.5 + 0.5 - \mu * 0.5 = 0.5$$

**Hence we see that  $\lambda = 0.5$  makes the total error probability independent of  $\mu$ .**

### 3 Linear Regression - Question 5, 6 and 7

In these problems, we will explore how Linear Regression for classification works. As with the Perceptron Learning Algorithm in Homework 1, you will create your own target function  $f$  and data set  $D$ . Take  $d = 2$  so you can visualize the problem, and assume  $X = [1, 1] \times [1, 1]$  with uniform probability of picking each  $x \in X$ . In each run, choose a random line in the plane as your target function  $f$  (do this by taking two random, uniformly distributed points in  $[1, 1] \times [1, 1]$  and taking the line passing through them), where one side of the line maps to  $+1$  and the other maps to  $-1$ . Choose the inputs  $x_n$  of the data set as random points (uniformly in  $X$ ), and evaluate the target function on each  $x_n$  to get the corresponding output  $y_n$ .

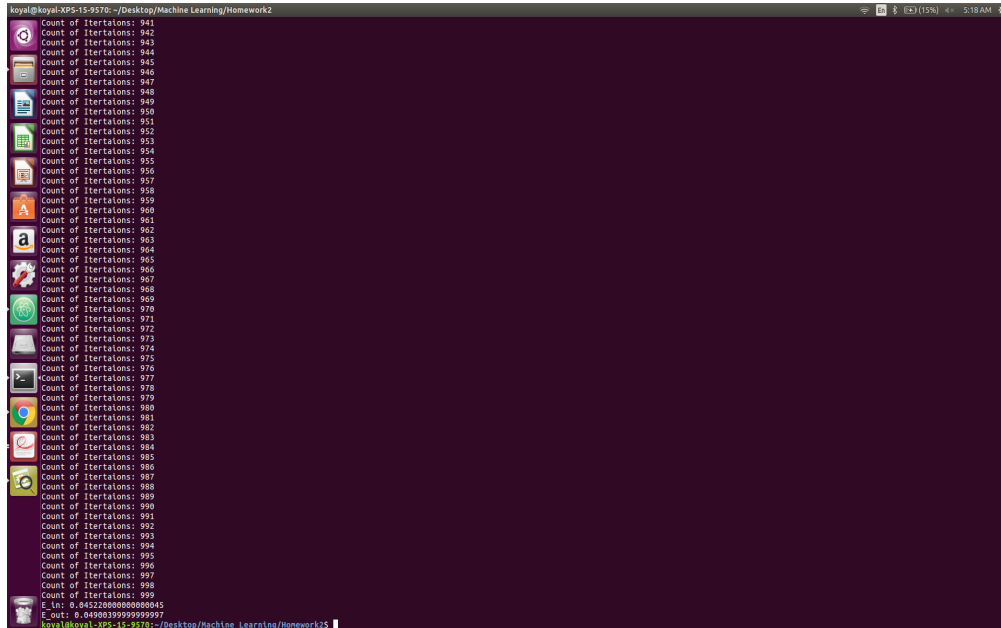


Figure 2: Ein and Eout while using Linear Regression for Classification

Please find the code attached for Question 5,6 in Section 7 - Linear Regression for Classification

#### 3.1 Question 5

Take  $N = 100$ . Use Linear Regression to find  $g$  and evaluate  $E_{in}$ , the fraction of in-sample points which got classified incorrectly. Repeat the experiment 1000 times and take the average (keep the  $g$ 's as they will be used again in Problem 6). Which of the following values is closest to the average  $E_{in}$ ? (Closest is the option that makes the expression —your answer— given option— closest to 0. Use this definition of closest here and throughout.) [a] 0 [b] 0.001 [c] 0.01 [d] 0.1 [e] 0.5

**ANSWER:[C]**

In this problem, linear regression is implemented by taking a random target function and accordingly classifying the the randomly generated data points.

The initial weights is got by taking pseudo inverse of the data set as shown below:

$$W = (X^T X)^{-1} X^T Y \quad (2)$$

where  $X=(1,x_1,x_2)$ . Hence we get  $w$ (weights) as a  $(3 \times 1)$  matrix.

The error or the no of misclassified points is averaged over the total no of iterations i.e 1000 Hence the in sample error as seen from the screenshot is **0.045**. This is closest to 0.01 from the given options. **Hence the answer is 0.01.**

### 3.2 Question 6

Now, generate 1000 fresh points and use them to estimate the out-of-sample error  $E$  out of  $g$  that you got in Problem 5 (number of misclassified out-of-sample points / total number of out-of-sample points). Again, run the experiment 1000 times and take the average. Which value is closest to the average  $E$  out ?

**ANSWER[C]**

The same code is used to find the value of  $E_{out}$  with the testing set as 1000 points. The same value of hypotheses that was used to find  $E_{in}$  is used to find  $E_{out}$ . This is generated in the same way by counting the no of misclassified points and averaging over the iterations.

As seen from the figure above, we get the value of  $E_{out}$  as **0.049**. This is closest to 0.01 from among the given options. **Hence the answer is 0.01.**

### 3.3 Question 7

**Please find the code attached for Question 7 in Section 8 - Linear Regression for PLA**

Now, take  $N = 10$ . After finding the weights using Linear Regression, use them as a vector of initial weights for the Perceptron Learning Algorithm. Run PLA until it converges to a final vector of weights that completely separates all the in-sample points. Among the choices below, what is the closest value to the average number of iterations (over 1000 runs) that PLA takes to converge? (When implementing PLA, have the algorithm choose a point randomly from the set of misclassified points at each iteration) a] 1 [b] 15 [c] 300 [d] 5000 [e] 10000

**ANSWER:[A]**

```

koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework2
/usr/bin/python: No module named virtualenvwrapper
virtualenvwrapper.sh: There was a problem running the initialization hooks.

If Python could not import the module virtualenvwrapper.hook_loader,
check that virtualenvwrapper has been installed for
VIRTUALENVWRAPPER_PYTHON=/usr/bin/python and that PATH is
set properly.
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework2/
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework2$ python3 LinReg_PLA.py
Used as the initial weight: [[-0.41397523 -1.09886405  0.85180026]]
iterations count: 1

```

Figure 3: PLA using the weights found using Linear Regression

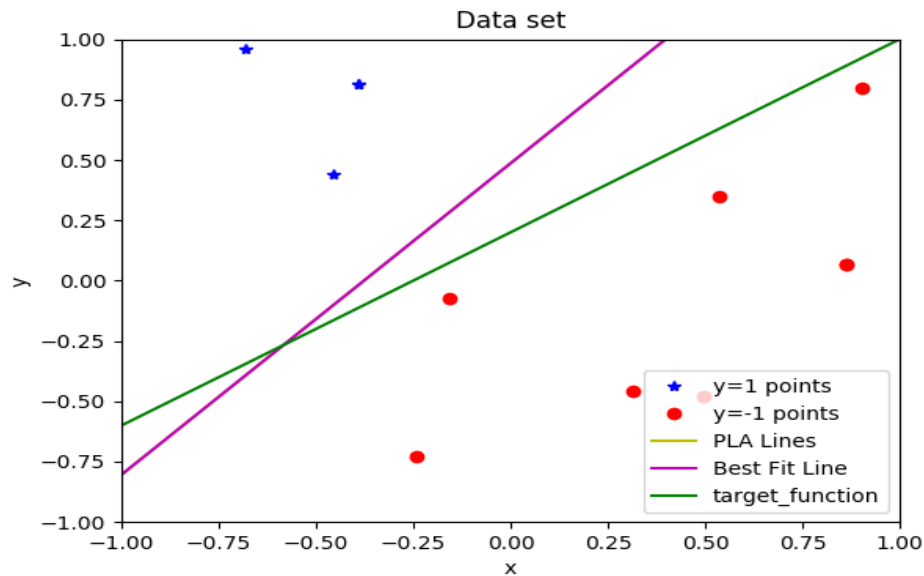


Figure 4: O/P showing the no of iterations that PLA takes to converge

The weights using Linear Regression calculated above is used as the initial weights for the PLA. This is done using a dataset of 10 points. As clearly seen the PLA converges in just 1 iteration. The screenshot of the terminal shows the weight that was calculated using the technique of pseudo inverse of the dataset. The value of the weights which leads to just 1 iteration to converge the PLA is :  $\mathbf{w}=[-0.41,-1.09,0.85]$

## 4 Nonlinear Transformation - Question 8, 9 and 10

In these problems, we again apply Linear Regression for classification. Consider the target function:  $f(x_1, x_2) = \text{sign}(x_1 + x_2 - 0.6)$ . Generate a training set of  $N = 1000$  points on  $X = [-1, 1] \times [-1, 1]$  with a uniform probability of picking each  $x \in X$ . Generate simulated noise by flipping the sign of the output in a randomly selected 10 percent subset of the generated training set.

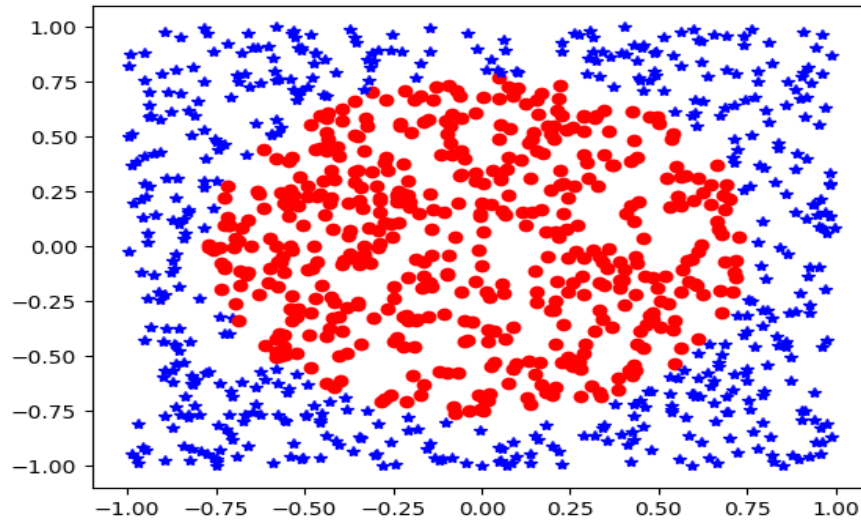


Figure 5: The given target function

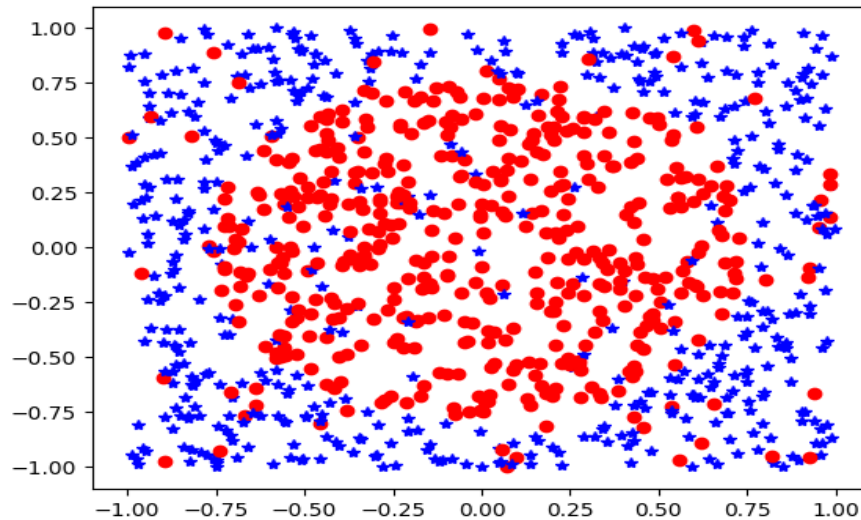


Figure 6: Target function with the simulated noise of 10 percent

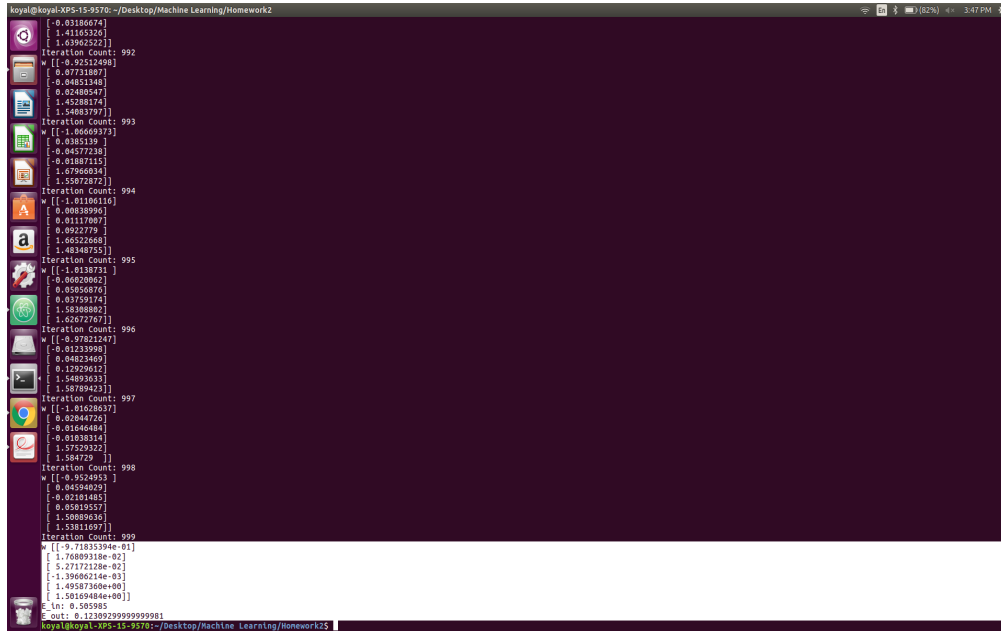


Figure 7: Target function with the simulated noise

Please find the code attached for Question 8,9,10 in Section 9 - Non Linear Transformation

#### 4.1 Question 8

Carry out Linear Regression without transformation, i.e., with feature vector:  $(1, x_1, x_2)$ , to find the weight  $w$ . What is the closest value to the classification in-sample error  $E_{in}$ ? (Run the experiment 1000 times and take the average  $E_{in}$  to reduce variation in your results.) [a] 0 [b] 0.1 [c] 0.3 [d] 0.5 [e] 0.8

**ANSWER[D]**

$E_{in}$  is calculated for the above training set of 1000 points. The iteration is run 1000 times to get a good approximation. As seen from the screenshot the  $E_{in}$  obtained is: **0.5**. This is also intuitively correct. The error is calculated by averaging the misclassified points on the total iterations. **Hence the answer is 0.5**

#### 4.2 Question 9

Now, transform the  $N = 1000$  training data into the following nonlinear feature vector:  $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ . Find the vector  $w$  that corresponds to the solution of Linear Regression. Which of the following hypotheses is closest to the one you find? Closest here means agrees the most with your hypothesis (has the highest probability of agreeing on a randomly selected point). Average over a few runs to make sure your answer is stable.

**ANSWER[A]**

The calculation of weights on the training data is done using the feature vector:  $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ . This is averaged over 10 runs. It is found using the pseudo inverse of the vector. The value of  $w$  so obtained is as shown:  $w = [-0.9, 0.01, 0.05, -0.003, 1.49, 1.5]$ . From the given options, these weights are closest to the following coefficients:  $[-1, -0.05, 0.08, 0.13, 1.5, 1.5]$ . Hence the answer is the following hypothesis vector:  $g(x_1, x_2) = \text{sign}(10.05x_1 + 0.08x_2 + 0.13x_1x_2 + 1.5x_1^2 + 1.5x_2^2)$



### 4.3 Question 10

What is the closest value to the classification out-of-sample error  $E$  out of your hypothesis from Problem 9? (Estimate it by generating a new set of 1000 points and adding noise, as before. Average over 1000 runs to reduce the variation in your results.) [a] 0 [b] 0.1 [c] 0.3 [d] 0.5 [e] 0.8

**ANSWER:[B]**

The above hypothesis which is averaged over 10 runs, is used to generate the out of sample error for a set of 1000 data points which is averaged over 1000 runs. The no of misclassified points using the weights calculated above is used to get the out of sample error as **0.123**. This value is closest to 0.1. **Hence the answer is 0.1**

## 5 Problem 3.9

Assuming  $X^T X$  is invertible show by direct comparison with eqn(3.4) that  $E_{in}(w)$  can be written as

$$E_{in}(w) = (w - (X^T X)^{-1} X^T y)^T (X^T X) (w - (X^T X)^{-1} X^T y) + y^T (I - X(X^T X)^{-1} X^T) y \quad (3)$$

. Use this expression to obtain  $w_{lin}$ . What is the in-sample error?

**Proof:**

Equation 3.4 is

$$E_{in}(w) = \frac{1}{N} (w^T X^T X w - 2w^T X^T y + y^T y) \quad (4)$$

$2w^T X^T y$  can be split as follows:

$$E_{in}(w) = \frac{1}{N} (w^T X^T X w - w^T X^T y - w^T X^T y + y^T y) \quad (5)$$

Which can also be written as follows as  $E_{in}$  is a scalar so all terms are constant and  $const = const^T$

$$E_{in}(w) = \frac{1}{N} (w^T X^T X w - y^T X w - w^T X^T y + y^T y) \quad (6)$$

Taking common terms and simplifying we get:

$$E_{in}(w) = \frac{1}{N} ((w^T X^T - y^T) X w - (w^T X^T - y^T) y) \quad (7)$$

$$E_{in}(w) = \frac{1}{N} ((w^T X^T - y^T) (X w - y)) \quad (8)$$

$$E_{in}(w) = \frac{1}{N} ((w X - y)^T (X w - y)) \quad (9)$$

As it is given that  $(X)X^T X$  is invertible, the following derivation holds:

$$(X^T X)^{-1}(X^T X) = I$$

Pre multiply by  $X$  and post multiply by  $X^T$  on both sides

$$(X)(X^T X)^{-1}(X^T X)X^T = X I X^T$$

Post multiplying both sides by  $(X X^T)^{-1}$

$$(X)(X^T X)^{-1}X^T(X X^T)(X X^T)^{-1} = X X^T(X X^T)^{-1}$$

Thus:

$$X(X^T X)^{-1}(X^T) = I \quad (10)$$

Putting this back in eqn 9 we get:

$$E_{in}(w) = \frac{1}{N}((wX - y)^T X(X^T X)^{-1}(X^T)(Xw - y)) \quad (11)$$

Opening the  $(Xw - y)$  term we get:

$$E_{in}(w) = \frac{1}{N}((wX - y)^T(X)((X^T X)^{-1}(X^T X)w - (X^T X)^{-1}(X^T y))) \quad (12)$$

$(X^T X)$  is invertible, hence:

$$E_{in}(w) = \frac{1}{N}((wX - y)^T(X)(w - (X^T X)^{-1}X^T y)) \quad (13)$$

Now considering the term:  $(wX - y)^T$  following derivation can be made:

$$(wX - y)^T \text{ is } (X^T w^T - y^T)$$

Using eqn(10) we have:  $(X^T)(X X^T)^{-1}(X) = I$

Hence:  $(X^T)(X X^T)^{-1}(X)(X^T w^T - y^T)$

$$(X^T)((X X^T)^{-1}(X X^T)w^T - (X X^T)^{-1}(X)y^T)$$

$$(X^T)(w^T - (X X^T)^{-1}(X)y^T)$$

Taking the transpose out:

$$((w^T - (X X^T)^{-1}(X)y^T)^T(X^T)^T)^T \text{ Thus:}$$

$$(wX - y)^T = ((w - (X^T X)^{-1}(X^T)y)^T(X)^T) \quad (14)$$

Putting this value of  $(wX - y)^T$  back into the equation 13 we get.

$$E_{in}(w) = \frac{1}{N}((w - (X^T X)^{-1}(X^T)y)^T (X^T X)(w - (X^T X)^{-1}X^T y)) \quad (15)$$

Now, lets consider

$$Y^T(I - I)Y = 0 \text{ here I is identity}$$

Also, using eqn() we have:

$$X(X^T X)^{-1}(X^T) = I$$

So substituting this in the place of I in the above eqn we get:

$$Y^T(I - X(X^T X)^{-1}X^T)Y = 0$$

Putting this term in eqn 15 as a connstant 0 term, we have:

$$E_{in}(w) = \frac{1}{N}((w - (X^T X)^{-1}X^T y)^T (X^T X)(w - (X^T X)^{-1}X^T y) + Y^T(I - X(X^T X)^{-1}X^T)Y) \quad (16)$$

which can be written as:

$$E_{in}(w) = ((w - (X^T X)^{-1}(X^T)y)^T (X^T X)(w - (X^T X)^{-1}X^T y) + Y^T(I - X(X^T X)^{-1}X^T)Y) \quad (17)$$

**Hence Proved.**

## **PART 2:**

Let the pseudo inverse of X i.e  $(X^T X)^{-1}X^T$  be represented as K. Thus the above proved equation can be written as:

$$(w - Ky)^T (X^T X)(w - Ky) + y^T(I - XK)y \quad (18)$$

This has to be differentiated wrt w to get  $w_{lin}$

Using the following properties of gradient identities:

$$\nabla(w^T Aw) = (A + A^T)w \quad (19)$$

$$\nabla(w^T b) = b \quad (20)$$

Comparing this formula with our eqn for  $E_{in}$  the following comparision is clear:

$$w' = (w - Ky) \text{ and } A = X^T X$$

Thus differentiating wrt  $w$  we get:

$$(X^T X + (X^T X)^T)(w - Ky) = 0 \quad (21)$$

Which is the same as:  $2(X^T X)(w - Ky) = 0$

Pre-multiplying both sides with  $(X^T X)^{-1}$  we get:

$$2(X^T X)^{-1}(X^T X)(w - Ky) = 0$$

As  $(X^T X)$  is invertible, we have:  $(X^T X)^{-1}(X^T X) = I$

hence:  $2(w - Ky) = 0$

Therefore:  $w_{lin} = Ky$

Re substituting value of  $K$  we get:

$$w_{lin} = (X^T X)^{-1} X^T y \quad (22)$$

**Hence the above is the value of  $w_{lin}$ .**

Now recalculating  $E_{in}$  with the  $w_{lin}$  obtained above we have:

Substitute this in eqn 17 we have:

$$E_{in}(w) = ((X^T X)^{-1} X^T y - (X^T X)^{-1} (X^T) y)^T (X^T X) [(X^T X)^{-1} X^T y - (X^T X)^{-1} X^T y] + Y^T (I - X(X^T X)^{-1} X^T) Y \quad (23)$$

Which gives us:

$$E_{in}(w) = 0 \quad (24)$$

Thus we see that the in sample error  $E_{in}$  for any given data set to be equal to zero.

Note: Codes for each of the questions is attached below for reference.

## 6 Codes 1(Hoeffding Inequality using virtual coins)

```
#
# Copyright 2019 Harsh Kakashaniya ,Koyal Bhartia
# @file      Hoeffding_Inequality.py
# @author    Koyal Bhartia
# @date      03/06/2019
# @version   1.0
#
# @brief This is the code for Question 1,2 of Homework 2 from "Learning from Data"
#
# @Description This code proves the implementation of Hoeffding Inequality using
# a single bin of 1000 virtual fair coins
#
#Import statments
import numpy as np
import random as rd
import math

#Initializations
# Number of coins
NumCoins=1000
# Array of all the Coins where ith element represents ith coin
Coins=np.zeros((1,NumCoins))
# No. of flips of each coin
NumFlips=10
# No. of iterations
Iterations=100000
# 3D Array to record the independent coin flip of each coin
FlipsMat=np.zeros((NumFlips,NumCoins,Iterations),dtype=int)
# 3D Array to store the fraction of heads of each coin for each iteration
HeadFrac=np.zeros((1,NumCoins,Iterations),dtype=float)
# Let
# 0 -> Head
# 1 -> Tail
#Counter to keep track of the iterations
Itercount=0
TotalVmin=0
TotalV1=0
TotalVrand=0

#Probability of the 3 events - C1, Crand and Cmin
```

```

probCount1=0
probCountrand=0
probCountmin=0
# Variables to calculate Hoeffding bound
mu=0.5
epsilon=0.2
N=10 # Sample size

#Calculating the Hoeffding bound
x=-2*math.pow(epsilon,2)*N
Hbound=2*math.exp(x)

#Returns the fraction of heads out of the 10 flips for each coin for each iteration
def countFracHead(Coin, Iteration):
    HeadCount=0
    for i in range(NumFlips):
        if (FlipsMat[i, Coin, Iteration]==0):
            HeadCount+=1
    HeadFrac[0, Coin, Iteration]=HeadCount/NumFlips
    return HeadFrac

#Runs the experiment for the specified no of iterations
while(Itercount<Iterations):
    print("Count_of_Iterations:", Itercount)
    for coins in range(NumCoins):
        for flip in range(NumFlips):
            FlipsMat[flip, coins, Itercount]=rd.randint(0,1)

    #1st Coin
    c1=FlipsMat[:, 0, :]
    #Random coin position
    crandpos=rd.randint(0,999)
    crand=FlipsMat[:, crandpos, :]
    #Calculating the fraction of heads for each coin
    for i in range(NumCoins):
        HeadFrac=countFracHead(i, Itercount)
    #Coin with min freq of heads
    cmin=np.argmin(HeadFrac[0, :, Itercount])

    #Getting the fraction of heads for the 3 events (c1, crand and cmin)
    v1=HeadFrac[0, 0, Itercount]
    vrand=HeadFrac[0, crandpos, Itercount]
    vmin=HeadFrac[0, cmin, Itercount]

```

```

TotalVmin+=vmin
TotalV1+=v1
TotalVrand+=vrand

#Calculation if the distribution of above v satisfies the Hoeffding inequality
diff1=np.abs(v1-mu)
diffrand=np.abs(vrand-mu)
diffmin=np.abs(vmin-mu)

if(diff1>epsilon):
    probCount1+=1
if(diffrand>epsilon):
    probCountrand+=1
if(diffmin>epsilon):
    probCountmin+=1

Itercount+=1

#taking average over the no of iterations
AvgVmin=TotalVmin/Iterations
AvgV1=TotalV1/Iterations
AvgVrand=TotalVrand/Iterations
print("AvgVmin:",AvgVmin)
print("AvgV1:",AvgV1)
print("AvgVrand:",AvgVrand)

Prob1=probCount1/Iterations
Probrand=probCountrand/Iterations
Probmin=probCountmin/Iterations
print("Prob_of_Event_C1:",Prob1)
print("Prob_of_Event_Crand:",Probrand)
print("Prob_of_Event_Cmin:",Probmin)

print("Hoeffding_Bound:",Hbound)
if(Prob1<Hbound):
    print("c1_satisfies_Hoeffding_inequality")
if(Probrand<Hbound):
    print("crand_satisfies_Hoeffding_inequality")
if(Probmin<Hbound):
    print("cmin_satisfies_Hoeffding_inequality")

```

## 7 Codes 2(Linear regression for Classification)

```
#
# Copyright 2019 Harsh Kakashaniya ,Koyal Bhartia
# @file Lienar Regression.py
# @author Koyal Bhartia
# @date 03/06/2019
# @version 1.0
#
# @brief This is the code for Question 5,6 of Homework 2 from "Learning from Data"
#
# @Description This code explores how Linear Regression can be used for Classification.
# Linear Regression is used to evaluate Ein and Eout; taking a randomly generated
# target function on a random set
#Import statments
import argparse
import numpy as np
from numpy import linalg as LA
import math
import matplotlib.pyplot as plt
import random

#Initializations
N_train=100 # Ein
N_test=1000 # Eout
N_trials=1000
E_in=np.zeros((1,N_trials))
E_out=np.zeros((1,N_trials))

def generate_data(N):
    t_slope=0
    t_intercept=0
    data=np.column_stack((np.ones(N),np.ones(N),np.ones(N)))
    for i in range(len(data)):
        data[i,0]=random.uniform(-1,1)
        data[i,1]=random.uniform(-1,1)

    # Taking random point to calculate target function
    ranPt1=random.randint(0,N_train-1)
    t_slope=random.randint(-10,10)/10
    t_intercept=data[ranPt1,1]-t_slope*data[ranPt1,0]
    for l in range(len(data)):
        if (data[l,0]*t_slope+t_intercept<data[l,1]):
            data[l,2]=1
```



```

        else:
            data[1,2]=-1
    return data, t_slope, t_intercept

def signCheck(num):
    if (num>0):
        return 1
    else:
        return -1

#Classification using linear regression
def LinRegression():
    misclassify_Ein=0
    misclassify_Eout=0
    data,t_slope,t_intercept=generate_data(N_train)
    #Solving for the cost function by taking pseudo inverse of X
    X_data=np.column_stack((np.ones(len(data)),data[:,0],data[:,1]))
    Y_data=data[:,2]
    w0=np.matmul(np.transpose(X_data),X_data)
    w1=LA.inv(w0)
    w2=np.matmul(w1,np.transpose(X_data))
    w3=np.matmul(w2,Y_data)
    w4=np.array([w3])
    w=np.transpose(w4)
    for i in range(N_train):
        mul=np.array([np.matmul(X_data,w)])
        y_train=signCheck(float(mul[0,i]))
        if(y_train!=data[i,2]):
            misclassify_Ein+=1
    #Saving the slope and intercept to be used by the testing set
    stg=t_slope
    ctg=t_intercept

    # Estimating E_out with fresh data
    data,ty,gy=generate_data(N_test)
    X_test=np.column_stack((np.ones(N_test),data[:,0],data[:,1]))
    for i in range(N_test):
        if (data[i,0]*stg+ctg<data[i,1]):
            data[i,2]=1
        else:
            data[i,2]=-1
    for i in range(N_test):
        mul1=np.matmul(X_test,w)

```

```

        y_test=signCheck(float(mul1[i,0]))
        if(y_test!=data[i,2]):
            misclassify_Eout+=1

    return misclassify_Ein , misclassify_Eout

TotEin=0
TotEout=0
count=0
for k in range(N_trials):
    print("Count_of_Itertaions:",k)
    misclassify_Ein , misclassify_Eout=LinRegression()
    E_in[0,k]=misclassify_Ein/N_train
    TotEin+=E_in[0,k]
    E_out[0,k]=misclassify_Eout/N_test
    TotEout+=E_out[0,k]
E_in=TotEin/N_trials
E_out=TotEout/N_trials
print("E_in:",E_in)
print("E_out:",E_out)

```

## 8 Codes 3(Linear regression for PLA)

```
#
# Copyright 2019 Harsh Kakashaniya ,Koyal Bhartia
# @file Linear Regression-PLA.py
# @author Koyal Bhartia
# @date 03/06/2019
# @version 1.0
#
# @brief This is the code for Question 7 of Homework 2 from "Learning from Data"
#
# @Description This code weights uses Linear Regression, to get the vector of
#initial weights for the Perceptron Learning Algorithm. The PLA is ran
# until it converges to a final vector of weights that completely separates
#all the in-sample points.
#
#Import statments
import argparse
import numpy as np
import os, sys
from numpy import linalg as LA
import math
import pickle
import matplotlib.pyplot as plt
import random
N=10
## data generation
#-----
data1=np.column_stack((np.ones(N),np.ones(N),np.ones(N)))
target_fn_m=0.8
# keep value of c between -1<c<1 because we have points between -1 to 1
target_fn_c=0.2

for i in range(0,len(data1)):

    data1[i,0]=random.uniform(-1,1)
    data1[i,1]=random.uniform(-1,1)
    #print(data1)
    if (data1[i,0]*target_fn_m+target_fn_c<data1[i,1]):
        a=i
        data1[i,2]=1
    else:
        b=i
        data1[i,2]=-1
```

```

#print (data1)
#-----
#data plotting
count=0
break_point=1
for i in range(0,len(data1)):
    if(data1[i,2]==1):
        plt.plot(data1[i,0],data1[i,1], '*b')
    else:
        plt.plot(data1[i,0],data1[i,1], 'or')

plt.plot(data1[a,0],data1[a,1], '*b', label='y=1_points')
plt.plot(data1[b,0],data1[b,1], 'or', label='y=-1_points')

def misclassified(data,w):
    break_point=1
    def safe(num):
        if (num>0):
            return 1
        else:
            return -1
    X=np.column_stack((np.ones(len(data1),dtype=int),data1[:,0],data1[:,1]))
    w=w.transpose()
    misclassify=[]
    for i in range(0,len(data1)):
        mul=np.matmul(X,w)
        sign=safe(float(mul[i,0]))
        if(sign!=data1[i,2]):
            misclassify=np.append([misclassify],[i])
    if(len(misclassify)==0):
        break_point = 0
        a=-5
        length=0
    if(break_point == 1):
        point=random.randint(1,len(misclassify))
        a=int(misclassify[point-1])
    length=len(misclassify)
    return a,break_point,length

X=np.column_stack((np.ones(len(data1),dtype=int),data1[:,0],data1[:,1]))

mis_counter=0

```

```

mis_avg=0
Y=data1[:,2]

#Using linear regression to calculate the vector of initial weights for the PLA
# algorithm
w0=np.matmul(np.transpose(X),X)
w1=LA.inv(w0)
w2=np.matmul(w1,np.transpose(X))
w3=np.matmul(w2,Y)
w=np.array([w3])
print("w_used_as_the_initial_weight",w)

while(break_point==1):
    count=count+1
    print("Iterations_count:",count)
    koyal,break_point,length=misclassified(data1,w)
    if(koyal!=-5):
        mis_counter=mis_counter+length
        w_new=np.mat([0,0,0])
        w_new= w + X[koyal,:]*data1[koyal,2]
        w=w_new
        m=float(-w_new[0,1]/w_new[0,2])
        c=float(-w_new[0,0]/w_new[0,2])
        x_line1=np.mat([[ -1],[1]])
        y_line1=m*x_line1+c
        plt.plot(x_line1,y_line1,'-y')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.axis([-1,1,-1,1])
        plt.title('Data_set')
        plt.pause(0.01)

    if(count==1):
        w_new=w

m=float(-w_new[0,1]/w_new[0,2])
c=float(-w_new[0,0]/w_new[0,2])
x_line1=np.mat([[ -1],[1]])
y_line1=m*x_line1+c
x_linet=x_line1
y_linet=target_fn_m*x_linet+target_fn_c

```

```
plt.plot(x_line1, y_line1, '-y', label='PLA_Lines')
plt.plot(x_line1, y_line1, '-m', label='Best_Fit_Line')
plt.plot(x_linet, y_linet, '-g', label='target_function')

plt.xlabel('x')
plt.ylabel('y')
plt.axis([-1,1,-1,1])
plt.title('Data_set')
plt.legend()
plt.show()
```

## 9 Codes 4(Nonlinear Transformation)

```
#
# Copyright 2019 Harsh Kakashaniya ,Koyal Bhartia
# @file Nonlinear Transformation.py
# @author Koyal Bhartia
# @date 03/06/2019
# @version 1.0
#
# @brief This is the code for Question 8,9,10 of Homework 2 from "Learning from Data"
#
# @Description This code applies Linear Regression for Classification on a
# Nonlinear function. The function is simulated with noise; and calcuation of
# Ein and Eout is made using the weights of a Nonlinear feature vector
#
#Import statments
import argparse
import numpy as np
import os, sys
from numpy import linalg as LA
import math
import pickle
import matplotlib.pyplot as plt
import random

#Initializations
N_train=1000
N_test=1000
N_trials=1000
E_in=np.zeros((1,N_trials))
E_out=np.zeros((1,N_trials))
noise_percent=0.1
wtot=np.mat(np.zeros((6,10)))

#Generation of random data
def generate_data(N):
    t_slope=0
    t_intercept=0
    X_data=np.column_stack((np.ones(N),np.ones(N)))
    fx=np.zeros((N))
    for i in range(len(X_data)):
        X_data[i,0]=random.uniform(-1,1)
        X_data[i,1]=random.uniform(-1,1)
        if (signCheck((np.power(X_data[i,0],2)+np.power(X_data[i,1],2)-0.6))==1):
```

```

        fx[i]=1
    else:
        fx[i]=-1
    return X_data, fx

#Adding noise to data
def add_noise(fx,N,noise_percent):
    Num_NoiseSample=int(N*noise_percent)
    randSample=random.sample(range(0, N-1), Num_NoiseSample)
    for i in range(Num_NoiseSample):
        if(fx[randSample[i]]==1):
            fx[randSample[i]]=-1
        else:
            fx[randSample[i]]=1
    return fx

def signCheck(num):
    if (num>0):
        return 1
    else:
        return -1

#Classification using linear regression
# Calcualtion of Ein
def LinRegression():
    misclassify=0
    misclassify1=0
    X_data,fx=generate_data(N_train)

    #Solving for the cost function by taking pseudo inverse of X
    X_data=np.column_stack((np.ones(len(X_data)),X_data[:,0],X_data[:,1]))
    w0=np.matmul(np.transpose(X_data),X_data)
    w1=LA.inv(w0)
    w2=np.matmul(w1,np.transpose(X_data))
    w3=np.matmul(w2,fx)
    w4=np.array([w3])
    w=np.transpose(w4)
    for i in range(N_train):
        mul=np.array([np.matmul(X_data,w)])
        y_train=signCheck(mul[0,i])
        if(y_train!=fx[i]):
            misclassify+=1
    return misclassify

```



*#calcualtion of Eout using the weights using non linear feature vector*

```

def LinRegression2():
    misclassify1=0
    X_data,fx=generate_data(N_train)
    fx=add_noise(fx,N_train,noise_percent)
    N=len(X_data)
    X_new=np.column_stack((np.ones(N),X_data[:,0],X_data[:,1],np.ones(N),np.ones(N),np.ones(N)))

    for i in range(N):
        X_new[i,3]=X_data[i,0]*X_data[i,1]
        X_new[i,4]=np.power(X_data[i,0],2)
        X_new[i,5]=np.power(X_data[i,1],2)

    for t in range(10):
        w0=np.matmul(np.transpose(X_new),X_new)
        w1=LA.inv(w0)
        w2=np.matmul(w1,np.transpose(X_new))
        w3=np.matmul(w2,fx)
        w4=np.array([w3])
        w=np.transpose(w4)
        wtot[:,t]=w
    w_new=wtot.mean(1)
    print("w",w_new)

    for i in range(N_test):
        mul1=np.matmul(X_new,w_new)
        y_test=signCheck(float(mul1[i,0]))
        if(y_test!=fx[i]):
            misclassify1+=1
    return misclassify1

TotEin=0
TotEout=0
count=0
for k in range(N_trials):
    print("Iteration_Count:",k)
    misclassify=LinRegression()
    misclassify1=LinRegression2()
    E_in[0,k]=misclassify/N_train
    TotEin+=E_in[0,k]
    E_out[0,k]=misclassify1/N_test

```

```

TotEout+=E_out[0,k]

#Plotting the function and the noise injected function
X_data,fx=generate_data(N_train)
for i in range(0,len(X_data)):
    if (fx[i]==1):
        plt.plot(X_data[i,0],X_data[i,1], '*b')
    else:
        plt.plot(X_data[i,0],X_data[i,1], 'or')
plt.show()
fx=add_noise(fx,N_train,noise_percent)
for i in range(0,len(X_data)):
    if (fx[i]==1):
        plt.plot(X_data[i,0],X_data[i,1], '*b')
    else:
        plt.plot(X_data[i,0],X_data[i,1], 'or')
plt.show()
E_in=TotEin/N_trials
E_out=TotEout/N_trials
print("E_in:",E_in)
print("E_out:",E_out)

```