Machine Learning (ENME808E): Homework 4

Koyal Bhartia(116350990)

April 16, 2019

1 Exercise 3.4 - Page 87 from LFD

It is given that the noisy target is $y = w^{*T}x + \epsilon$

1.1 Show that in-sample estimate of y is given by $\hat{y} = Xw^* + H\epsilon$

To prove this we use that known relation between y and $\hat{y}$ i.e $\hat{y} = Hy$ where $H = X(X^TX)^-1X^T$.

Using this we can re write $\hat{y}$ as

$$\hat{y} = H(w^{*T}X + \epsilon) \tag{1}$$

$\Rightarrow \hat{y} = Hw^{*T}X + H\epsilon$

Using the expression of H stated above we get:

$$\hat{y} = (X(X^TX)^{-1}X^T)w^{*T}X + H\epsilon \tag{2}$$

which is equal to $\hat{y} = (XX^{-1}(X^T)^{-1}X^T)w^{*T}X + H\epsilon$ or $\hat{y} = (I)w^{*T}X + H\epsilon$;

According to matrix multiplication we have $w^{*T}X$=Xw^*

Thus the above equation is:

$$\hat{y} = (I)w^{*T}X + H\epsilon \tag{3}$$

1.2 Show that the in-sample error vector $\hat{y} - y$ can be expressed as a matrix times *epsilon*. What is the matrix

Using the expression of y and $\hat{y}$ i.e $y = w^{*T}x + \epsilon$ and $\hat{y} = Xw^* + H\epsilon$ we get:

$$\hat{y} - y = Xw^* + H\epsilon - w^{*T}x + \epsilon \tag{4}$$

As Xw^* and $w^{*T}x$ results in similar matrices, it can be cancelled. Hence:

$$\hat{y} - y = (H - I)\epsilon \tag{5}$$

1.3 Express $E_{in}(w_{lin})$ in terms of ϵ

It is known that $E_{in}(w_{lin})$ can be expressed as:

$$E_{in}(\mathbf{w}_{lin}) = \frac{1}{N}(\hat{\mathbf{y}} - \mathbf{y})^T(\hat{\mathbf{y}} - \mathbf{y}) \tag{6}$$

Using the expression for $\hat{y} - y$ calculated above, eqn 24 can be written as:

$$E_{in}(\mathbf{w}_{lin}) = \frac{1}{N}(\epsilon^T(H - I)^T(H - I)\epsilon) \tag{7}$$

This is the same as:

$$E_{in}(\mathbf{w}_{lin}) = \frac{1}{N}(\epsilon^T(I - H)^T(I - H)\epsilon) \tag{8}$$

Using the expression $(I - H)^K = (I - H)$, we get:

$$E_{in}(\mathbf{w}_{lin}) = \frac{1}{N}\epsilon^T(I - H)\epsilon = \frac{1}{N}\epsilon^T\epsilon - \frac{1}{N}\epsilon^T H\epsilon \tag{9}$$

Hence above is the expression of $E_{in}(w_{lin})$ in terms of ϵ.

1.4 Prove that $E_D[E_{in}(w_{lin})] = \sigma^2(1 - \frac{d+1}{N})$

Here we just need to find the expected value of $E_{in}(w_{lin})$ obtained above. We can split it as expected value of each of the individual terms. i.e

$$\mathbb{E}_D[E_{in}(\mathbf{w}_{lin})] = \mathbb{E}_D[\frac{1}{N}\epsilon^T\epsilon] - \mathbb{E}_D[\frac{1}{N}\epsilon^T H\epsilon] \tag{10}$$

Solving for $\mathbb{E}_D[\frac{1}{N}\epsilon^T\epsilon]$: It is given that ϵ has 0 mean, and condidering gaussian noise. Hence $\epsilon^T\epsilon = \sigma^2$.

Thus:

$$\mathbb{E}_D[\frac{1}{N}\epsilon^T\epsilon] = \frac{N\sigma^2}{N} = \sigma^2 \tag{11}$$

Now solving for $\mathbb{E}_D[\frac{1}{N}\epsilon^T H\epsilon]$: We have:

$$\frac{1}{N}\epsilon^T H\epsilon = \frac{1}{N}\{\sum_{i=1}^{N} H_{ii}\epsilon_i^2 + \sum_{i,j=1}^{N} H_{ij}\ \epsilon_i\ \epsilon_j\} \tag{12}$$

Here H_{ii} are the diagonal terms. It is given that the trace(H)=d+1. Using this and eqn 25, we get:

Using trace(H)=d+1, clearly:

$$\frac{1}{N}\mathbb{E}_D[\sum_{i=1}^{N} H_{ii}\epsilon_i^2] = \frac{1}{N}(trace(H)\ \sigma^2) = \frac{1}{N}(d+1)\ \sigma^2 \tag{13}$$

Now considering the H_{ij} terms: Assume something like a gaussian error curve, and the error

componenets here i.e ϵ_i and ϵ_j are independent from each other, plus they also have 0 mean, hence all the terms will cancel out each other. Thus this implies:

$$\frac{1}{N}\mathbb{E}_D[\sum_{i,j=1}^{N} H_{ij}\ \epsilon_i\ \epsilon_j] = 0 \tag{14}$$

Plugging in eqn 13 and eqn 14 and eqn 11 in eqn 10, we have:

$$\mathbb{E}_D[E_{in}(\mathbf{w}_{lin})] = \sigma^2(1 - \frac{d+1}{N}) \tag{15}$$

1.5 Prove that $\mathbb{E}_{D,\epsilon'}[E_{test}(w_{lin})] = \sigma^2\left(1 + \frac{d+1}{N}\right)$

Here the training set and the final hypothesis remains the same as the above case. i.e $y = Xw^* + \epsilon$ The test data set here is given by: $y^{'} = Xw^* + \epsilon^{'}$

Here we follow the steps as above to find the expected value for the testing set. Hence here again we find: $\hat{y} - y$ which we get as $H\epsilon - I\epsilon^{'}$. Again we have:

$$E_{test}(\mathbf{w}_{lin}) = \frac{1}{N}(\hat{\mathbf{y}} - \mathbf{y})^T(\hat{\mathbf{y}} - \mathbf{y}) \tag{16}$$

Thus here this becomes:

$$E_{test}(\mathbf{w}_{lin}) = \frac{1}{N}((H\epsilon - I\epsilon^{'})^T(H\epsilon - I\epsilon^{'})) \tag{17}$$

Splitting the above we get:

$$E_{test}(\mathbf{w}_{lin}) = \frac{1}{N}(\epsilon^T H^T - \epsilon^{'T})(H\epsilon - \epsilon^{'}) = \frac{1}{N}(\epsilon^T H\epsilon - \epsilon^T H\epsilon^{'} - \epsilon^{'T} H\epsilon + \epsilon' T\epsilon^{'}) \tag{18}$$

We have already proved in the previous part that:

$$\mathbb{E}_{D,\epsilon'}[\epsilon^T H\epsilon] = \sigma^2\frac{d+1}{N} \tag{19}$$

and..

$$\mathbb{E}_{D,\epsilon'}[\epsilon' T\epsilon^{'}] = \sigma^2 \tag{20}$$

and..

$$\mathbb{E}_{D,\epsilon'}[\epsilon^T H\epsilon^{'}] = \sum_{i=1}^{N}\epsilon_i H_{ij}\epsilon_j^{'} = 0 \tag{21}$$

$$\mathbb{E}_{D,\epsilon'}[\epsilon^{'T} H\epsilon] = \sum_{i=1}^{N}\epsilon_i^{'} H_{ij}\epsilon_j = 0 \tag{22}$$

Thus,

$$\mathbb{E}_{D,\epsilon'}[E_{test}(w_{lin})] = \sigma^2\left(1 + \frac{d+1}{N}\right) \tag{23}$$

2 Homework 5 Problems from LFD

Linear Regression Error

2.1 Question 1

ANSWER:[c]

It is given that for the case of noisy target $y = w^{*T}x + \epsilon$, it can be shown that the expected in-sample error E_{in} wrt D can be given by:

$$E_D[E_{in}(w_{lin})] = \sigma^2(1 - \frac{d+1}{N}) \tag{24}$$

Given: $\sigma = 0.1$, $d = 8$ To find: Smallest value of N if smallest value of $E_{in} = 0.008$

i.e

$$E_D[E_{in}(w_{lin})] \geq 0.008 \tag{25}$$

Hence from eqn 24 and 25, we have:

$$\sigma^2(1 - \frac{d+1}{N}) \geq 0.008 \tag{26}$$

$\Rightarrow (1 - \frac{d+1}{N}) \geq \frac{0.008}{\sigma^2}$

$\Rightarrow 1 - \frac{0.008}{\sigma^2} \geq \frac{d+1}{N}$

$\Rightarrow \frac{1}{d+1}(1 - \frac{0.008}{\sigma^2}) \geq \frac{1}{N} \Rightarrow \frac{1}{d+1}(\frac{\sigma^2 - 0.008}{\sigma^2}) \geq \frac{1}{N}$

Thus:

$$N \geq d + 1(\frac{\sigma^2}{\sigma^2 - 0.008}) \tag{27}$$

Now plugging in the given values of σ and d we get:

$$N \geq 8 + 1(\frac{0.1^2}{0.1^2 - 0.008}) \tag{28}$$

Solving the above we get $N \geq 45$.

Thus from the given options, the smallest no of examples that will result in E_{in} to be greater than 0.008 is 100. Hence the answer is option c.

Nonlinear Transforms

2.2 Question 2

ANSWER:[d]

Here the given transform is : $\Phi(1, x_1, x_2) = (1, x^1, x^2)$ The decision boundary in X corresponds to a hyperbola. The region "between" the 2 curves of the hyperbola is the desired boundary, i.e where the feature is +1.

Considering the eqn of a hyperbola i.e

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \tag{29}$$

Considering w1 to be the weight of x^2 and w2 to be the weight of y^2, the region on both sides of the hyperbola can be got with the condition $w1 > 0$ and $w2 < 0$.

But in the model given in the question, we need the region between the hyperbolic curves. For this we need both the x^2 and y^2 to be negavtive; this is possible only if $w1 < 0$ and $w2 > 0$. Thus the asnwer is option d.

2.3 Question 3

ANSWER:[c]

It is known that in the case of linear classification, the VC dimension is given by d+1; where d is the no of features. +1 is for the constant term.

In this case, the polynomial transform that is given is:

$$\Phi : x \rightarrow (1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3, x_1^4, x_1^3x_2, x_1^2x_2^2, x_1x_2^3, x_2^4) \tag{30}$$

Here on counting the no of distinct features above we get d=14.

Hence according to the definition the VC dimension will be 14+1 i.e 15.

Hence the answer is option c.

Gradient Descent

2.4 Question 4

ANSWER:[e]

It is given that $E(u, v) = (ue^v - 2ve^{-u})^2$

Differentiating this wrt u i.e $\frac{\partial E}{\partial u}$ we get:

$\frac{\partial E}{\partial u} = 2(ue^v - 2ve^{-u})\frac{\partial(ue^v - 2ve^{-u})}{\partial u}$

$\Rightarrow 2(ue^v - 2ve^{-u})(\frac{\partial(ue^v)}{\partial u} + \frac{\partial(-2ve^{-u})}{\partial u})$

$\Rightarrow 2(ue^v - 2ve^{-u})(e^v + 2ve^{-u}))$

The above derivative matches with option e from among the given options. Hence the answer is e.

2.5 Question 5

ANSWER:[d]

The partial derivative of E(u,v) wrt v i.e $\frac{\partial E}{\partial v}$ will also be needed to minimize the gradient descent.:

$\frac{\partial E}{\partial v} = 2(ue^v - 2ve^{-u})\frac{\partial(ue^v - 2ve^{-u})}{\partial v}$

$\Rightarrow 2(ue^v - 2ve^{-u})(\frac{\partial(ue^v)}{\partial v} + \frac{\partial(-2ve^{-u})}{\partial v})$

$\Rightarrow 2(ue^v - 2ve^{-u})(uve^v - 2e^{-u}))$

Now using $\eta = 0.1$ and the initial values of u,v as 1, we iteratively calculate the new values of u and v using the following form of gradient descent i.e

$$u = u - \eta(\frac{\partial E}{\partial u}) \tag{31}$$

$$v = v - \eta(\frac{\partial E}{\partial v}) \tag{32}$$

Following is the code snippet illustrating the same:

```python
import math
N=0
eta=0.1
u=1
v=1
Eu=u*math.exp(v)-2*v*math.exp(-u)
E=math.pow(Eu,2)
while E>math.pow(10,-14):
    print("N:",N)
    du=2*(math.exp(v)+2*v*math.exp(-u))*(u*math.exp(v)-2*v*math.exp(-u))
    dv=2*(u*math.exp(v)-2*v*math.exp(-u))*(u*math.exp(v)-2*math.exp(-u))
    u=u-eta*du
    v=v-eta*dv
    Eu=u*math.exp(v)-2*v*math.exp(-u)
    E=math.pow(Eu,2)
    print("E(u,v):",E)
    N+=1

print("Final N:",N)
print("Final u,v:",u,v)
```

Figure 1: Code of Ques 5,6

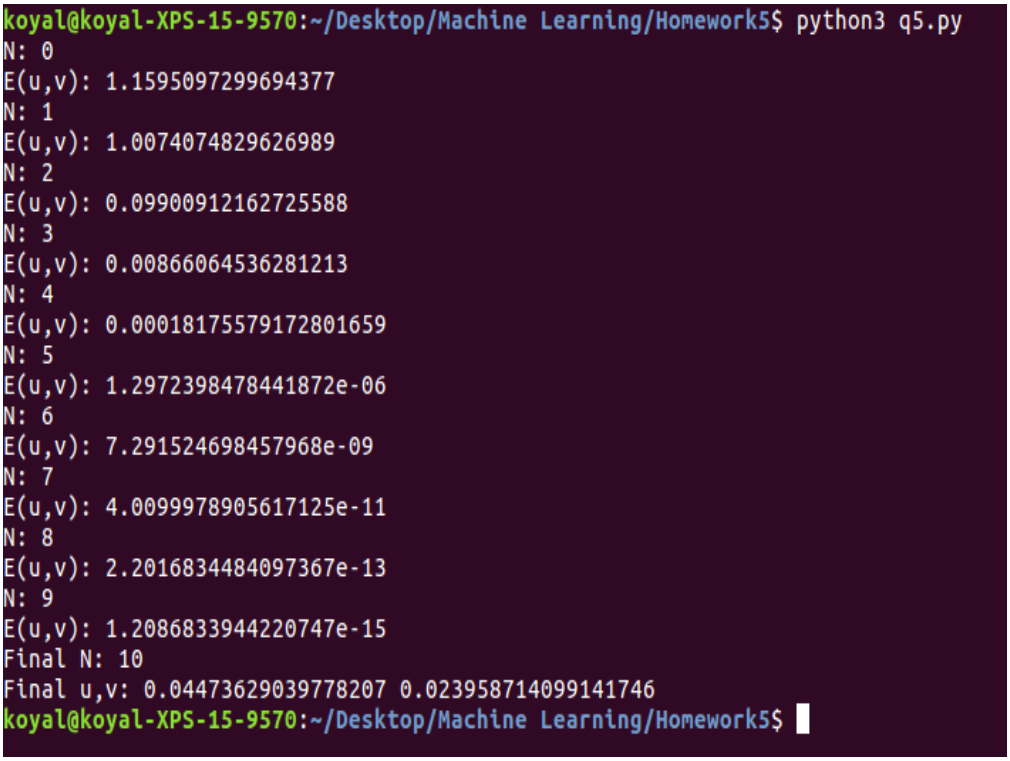


Figure 2: Screesnhot of o/p of Ques 5,6

Using the above updated values of u and v, E is updated using:

$$E(u,v) = (ue^{v} - 2ve^{-u})^2 \tag{33}$$

The above updating process is continuously carried out till the value of E falls below 10^{-14} for the first time.

As seen from the screenshot above, the no of iterations taken for E to drop below 10^{-14} is 10. Hence the answer is d.

2.6 Question 6

ANSWER:[e]

As seen from the above screenshot, the final values of u and v that drives E below 10^{-14} is 0.0447 and 0.0239 respectively.

This is closest to (0.045,0.024) from among the given options. Hence the answer is option e.

2.7 Question 7

ANSWER:[a]

Here the gradient descent is performed in 2 steps. We move along the u coordinate first reducing its error and then use that updated value of u to reduce the error along the v coordinate. This is repeated for 15 iterations, and then the Error(u,v) is checked.

Following is the code snippet illustrating the steps followed:

```
import math
N=0
eta=0.1
u=1
v=1
Eu=u*math.exp(v)-2*v*math.exp(-u)
E=math.pow(Eu,2)
while N<15:
    print("N:",N)
    du=2*(math.exp(v)+2*v*math.exp(-u))*(u*math.exp(v)-2*v*math.exp(-u))
    u=u-eta*du
    dv=2*(u*math.exp(v)-2*v*math.exp(-u))*(u*math.exp(v)-2*math.exp(-u))
    v=v-eta*dv
    Eu=u*math.exp(v)-2*v*math.exp(-u)
    E=math.pow(Eu,2)
    print("E(u,v):",E)
    N+=1

print("Final E:",E)
```

Figure 3: Code of o/p of Ques 7

Follwoing is the screenshot showing the value of E after each iteartion and the final E(u,v) after 15 iterations.

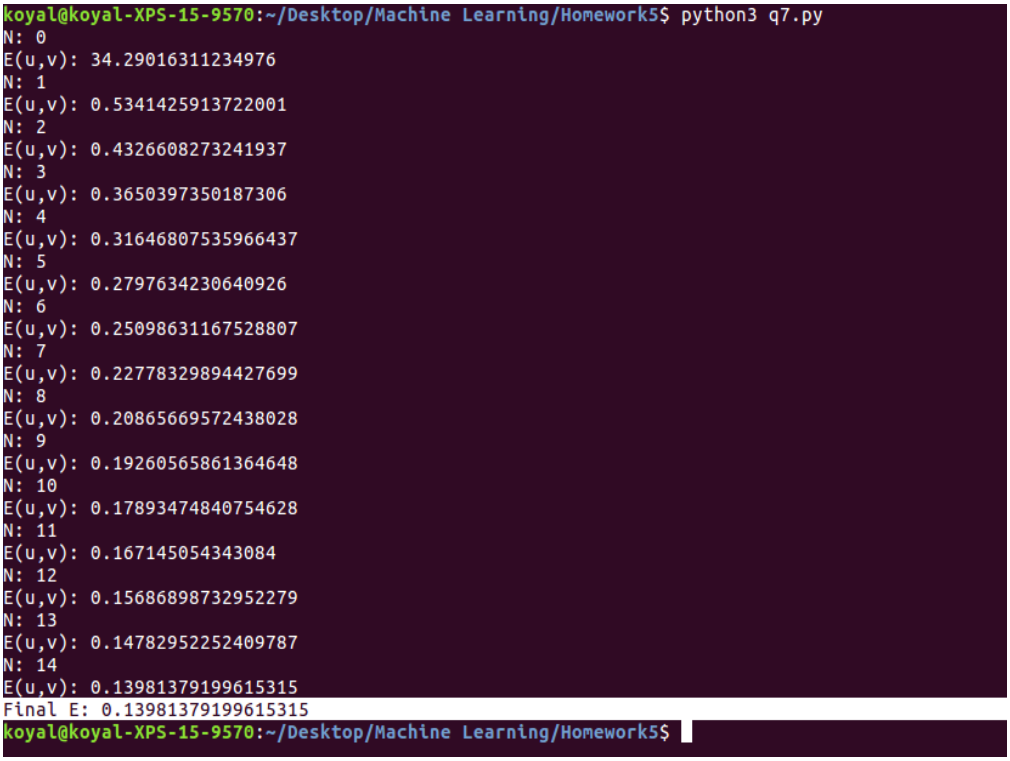


Figure 4: Screesnhot of o/p of Ques 7

We see that the value of E is 0.1398 which is closest to 10^{-1} from among the given options. Hence the asnwer is option a.

Logistic Regression

2.8 Question 8

ANSWER:[d]

Logistic Regression with Stochastic Gradient Descent has been implemented using the steps given in the question. The target function is a line joining any 2 random points from among the set of all random set of training points. The formula used to calculate E_{out} is:

$$E = log(1.0 + \exp -y * W^T x) \tag{34}$$

The above error is calculated over the testing data set using the final w that is received after the SGD algorithm converges. This error is averaged over 100 runs. w of every epoch is updated using:

$$w^{t+1} = w^t - \eta(-\frac{y_n x_n}{1 + \exp y_n w^{T(t)x_n}}) \tag{35}$$

where w^t is epoch at end of epoch t.

Following are 2 sample illustrations of the algorithm implemented: The code can be found in the codes section.

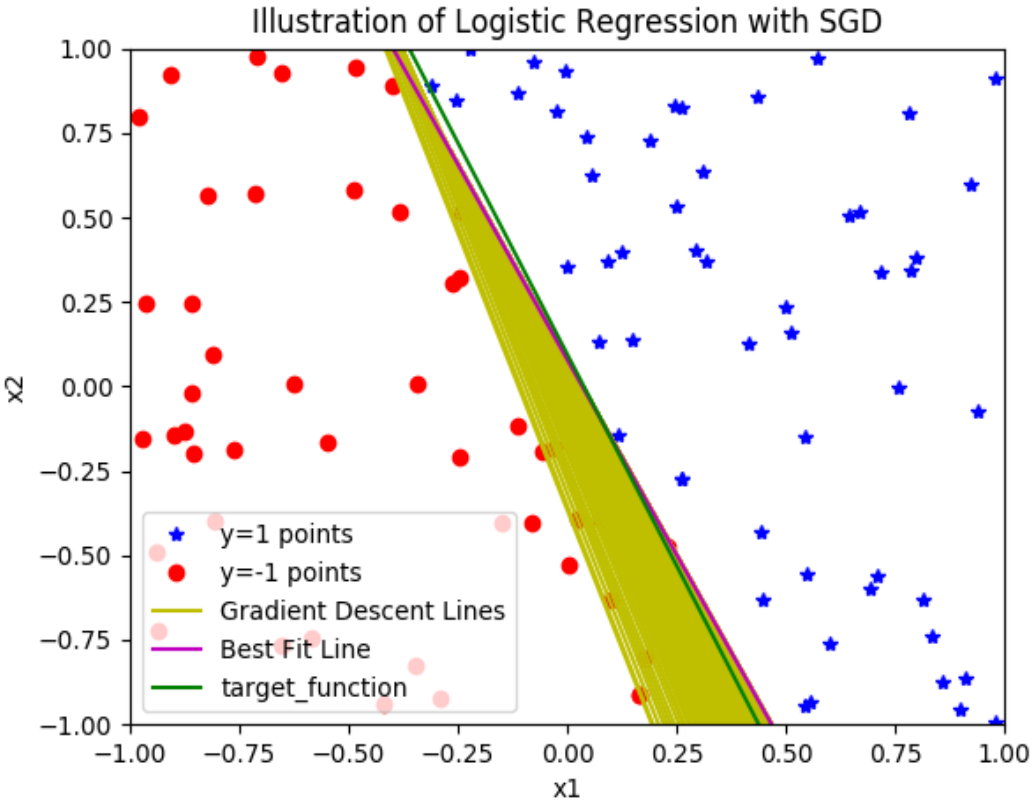
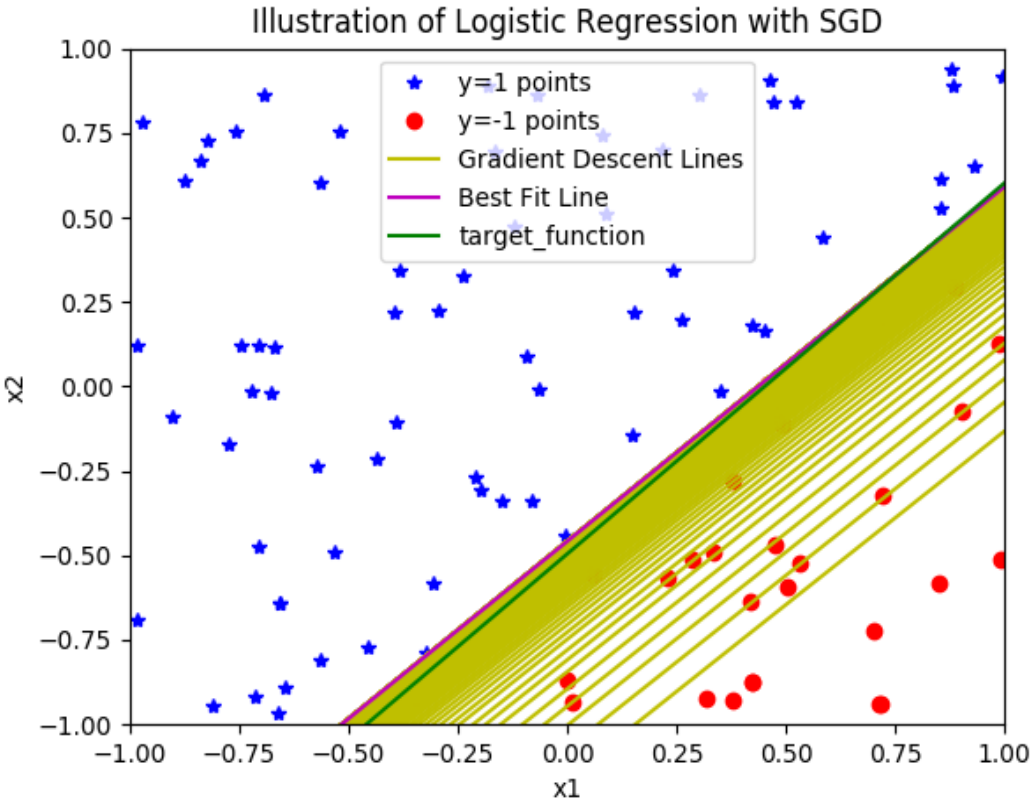


Illustration of Logistic Regression with SGD
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x1
y=1 points
y=-1 points
Gradient Descent Lines
Best Fit Line
target_function



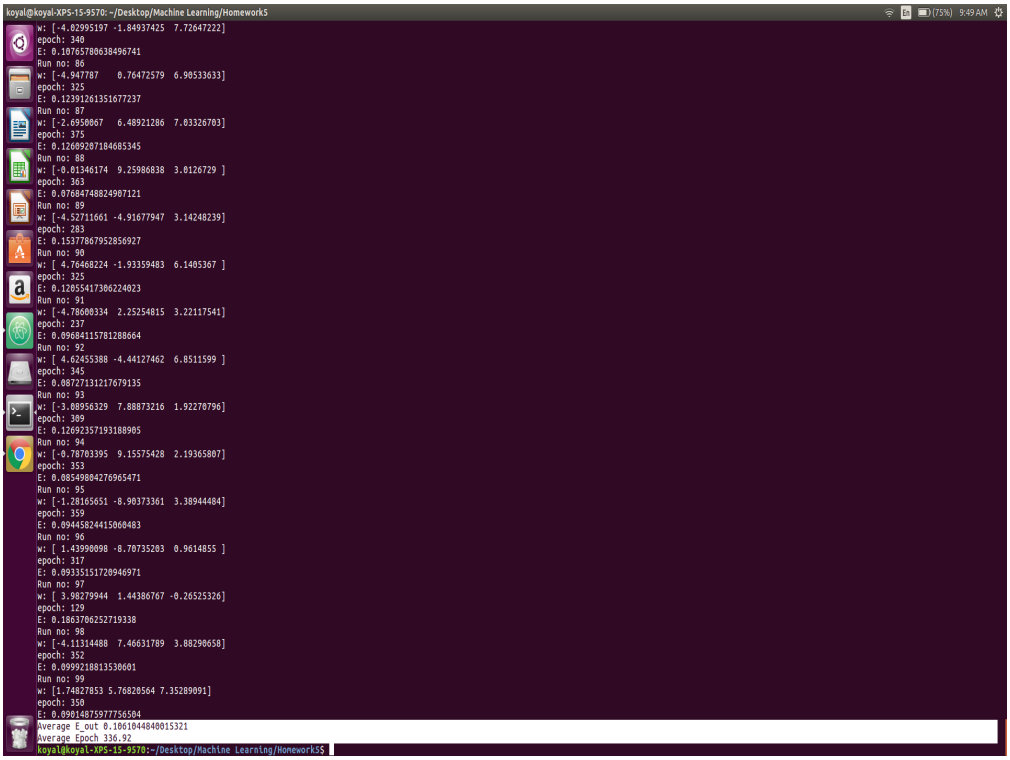


Figure 5: Screenshot of Output of Ques 8 and 9

As seen from the screenshot above, the average value of E_{out} we get is 0.1001. This is closest to 0.100 from among the given options. Hence the asnwer is option d.

2.9 Question 9

ANSWER:[a]

For each run, the no of epochs needed for the SGD to converge such that $||w(t-1) - w(t)|| < 0.01$ is recorded. This is averaged over 100 runs.

As seen from the scrreshot of the output, the average no of epochs is 336. This is closest to 350 from among the given options. Hence the answer is option a.

PLA as SGD

2.10 Question 10

ANSWER:[e] Stating what is known about the PLA algorithm:

$$w^{t+1} = w^t + y_n x_n for sign(w^T x_n) \neq y_n \tag{36}$$

and..

$$w^{t+1} = w^t + 0 for sign(w^T x_n) = y_n \tag{37}$$

Here x_n is known as the misclassified point.

In the case of SGD algorithm considered in the previous problem we updated the weights using the following:

$$w^{t+1} = w^t - \eta(-\frac{y_n x_n}{1 + \exp y_n w^{T(t) x_n}} \tag{38}$$

or..

$$w^{t+1} = w^t + y_n x_n(\frac{\eta}{1 + \exp y_n w^{T(t) x_n}} \tag{39}$$

The difference her is that PLA takes into account only misclassified points whereas SGD considers all the points.Thus from eqn 36n and 37, and comparing eqn 37 and eqn 39, we need to take the min of 0 and $y_n w^T x_n$ as we try to minimize the error function in the case of SGD.

Now min(0,$y_n w^T x_n$) will always be lesser than or equal to 0. Hence to get the proper error function we take the negative of the term. Thus PLA can be implemented as SGD with the error fuction $e_n(w) = -min(0, y_n w^T x_n)$. Thus the asnwer is option e.

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

3 Codes 1(Question 5,6 from Homework 5)

```
import math
N=0
eta=0.1
u=1
v=1
Eu=u*math.exp(v)-2*v*math.exp(-u)
E=math.pow(Eu,2)
while E>math.pow(10,-14):
    print("N:",N)
    du=2*(math.exp(v)+2*v*math.exp(-u))*(u*math.exp(v)-2*v*math.exp(-u))
    dv=2*(u*math.exp(v)-2*v*math.exp(-u))*(u*math.exp(v)-2*math.exp(-u))
    u=u-eta*du
    v=v-eta*dv
    Eu=u*math.exp(v)-2*v*math.exp(-u)
    E=math.pow(Eu,2)
    print("E(u,v):",E)
    N+=1

print("Final N:",N)
print("Final u,v:",u,v)
```

4 Codes 1(Question 7 from Homework 5)

```
import math
N=0
eta=0.1
u=1
v=1
Eu=u*math.exp(v)-2*v*math.exp(-u)
E=math.pow(Eu,2)
while N<15:
    print("N:",N)
    du=2*(math.exp(v)+2*v*math.exp(-u))*(u*math.exp(v)-2*v*math.exp(-u))
    u=u-eta*du
    dv=2*(u*math.exp(v)-2*v*math.exp(-u))*(u*math.exp(v)-2*math.exp(-u))
    v=v-eta*dv
    Eu=u*math.exp(v)-2*v*math.exp(-u)
    E=math.pow(Eu,2)
    print("E(u,v):",E)
    N+=1

print("Final E:",E)
```

5 Codes 1(Question 8,9 from Homework 5)

```
import argparse
import numpy as np
from numpy import linalg as LA
import math
import matplotlib.pyplot as plt
import random

def generate_data(N):
    X_data=np.column_stack((np.ones(N),np.ones(N)))
    fx=np.zeros((N))
    point1=random.randint(0,N-1)
    point2=random.randint(0,N-1)
    for i in range(len(X_data)):
        X_data[i,0]=random.uniform(-1,1)
        X_data[i,1]=random.uniform(-1,1)
    t_slope=(X_data[point1,1]-X_data[point2,1])/(X_data[point1,0]-X_data[point2,0])
    t_intercept=X_data[point1,1]-t_slope*X_data[point1,0]
    for i in range(len(X_data)):
        if (X_data[i,0]*t_slope+t_intercept<X_data[i,1]):
            a=i
            fx[i]=1
        else:
            b=i
            fx[i]=-1
    return X_data, fx,a,b,t_slope,t_intercept

def generate_test(N,t_slope,t_intercept):
    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 (X_data[i,0]*t_slope+t_intercept<X_data[i,1]):
            a=i
            fx[i]=1
        else:
            b=i
            fx[i]=-1
    return X_data, fx

def plotdata(X_data, fx,a,b):
    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.plot(X_data[a,0],X_data[a,1],'*b', label='y=1 points')
    plt.plot(X_data[b,0],X_data[b,1],'or', label='y=-1 points')


def LogitRegression(X_data,fx):
    epoch=0
    diff=1
    w=[0,0,0]
    w_prev=[0,0,0]
    X=np.column_stack((np.ones(N_train),X_data[:,0],X_data[:,1]))
    while(diff>=0.01):
        epoch+=1
        postion=random.sample(range(N_train),N_train)
        for i in range(N_train):
            delta=-(fx[postion[i]] * X[postion[i]])/(1.0 + math.exp(fx[postion[i]]*np.dot(w
            w=w-eta*delta
        m=float(-w[1]/w[2])
        c=float(-w[0]/w[2])
        x_line1=np.mat([[-1],[1]])
        y_line1=m*x_line1+c
        plt.plot(x_line1,y_line1,'-y')
        diff=np.sqrt(np.sum((w_prev - w)**2))
        w_prev=w
    return w, epoch

def Eout_Calc(X_data,fx,w):
    E_out=np.zeros((N_test))
    X=np.column_stack((np.ones(N_test),X_data[:,0],X_data[:,1]))
    for i in range(N_test):
        E_out[i] = np.log(1.0 + math.exp(-fx[i] * np.dot(w,X[i])))
    #print(np.shape(E_out))
    E=np.mean(E_out)
    return E


if __name__ == '__main__':
    N_train=100
    N_test=100
```

```
runs=100
w=[0,0,0]
eta=0.01
E_tot=0
E=0
Epoch_tot=0

for run in range(runs):
    print("Run no:",run)
    X_data,fx,a,b,t_slope,t_intercept=generate_data(N_train)
    plotdata(X_data,fx,a,b)
    w,epoch=LogitRegression(X_data,fx)
    Epoch_tot+=epoch
    print("w:",w)
    print("epoch:",epoch)

    X_data,fx=generate_test(N_test,t_slope,t_intercept)
    E=Eout_Calc(X_data,fx,w)
    print("E:",E)
    #print(np.shape(w))
    E_tot+=E
    #print()
    m=float(-w[1]/w[2])
    c=float(-w[0]/w[2])
    x_line1=np.mat([[-1],[1]])
    y_line1=m*x_line1+c
    y_linet=t_slope*x_line1+t_intercept

    plt.axis([-1,1,-1,1])
    plt.plot(x_line1,y_line1,'-y',label='Gradient Descent Lines')
    plt.plot(x_line1,y_line1,'-m',label='Best Fit Line')
    plt.plot(x_line1,y_linet,'-g',label='target_function')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title('Illustration of Logistic Regression with SGD')
    plt.legend()
    plt.show()

print("Average E_out",E_tot/runs)
print("Average Epoch",Epoch_tot/runs)
```