CS 156a - Problem Set 5

Samuel Patrone, 2140749

October 31, 2022

The following notebook is publicly available at the following link.

Contents

Problem 1

Answer: [c] 100

Derivation:

The expected value on a data set $\mathcal{D}$ of N samples of in-sample error for a noisy target function with variance σ^2 using linear regression in d dimension is

$$\mathbb{E}_{\mathcal{D}}[E_{in}] = \sigma^2 \left(1 - \frac{d+1}{N}\right) . \tag{1}$$

For $\sigma = 0.1$, $d = 8$ and $\mathbb{E}_{\mathcal{D}}[E_{in}] \geq 0.008$, we need at least

$$N \geq \frac{d+1}{\left(1 - \frac{\mathbb{E}_{\mathcal{D}}[E_{in}]}{\sigma^2}\right)} = \frac{9}{\left(1 - \frac{0.008}{(0.1)^2}\right)} = 45 . \tag{2}$$

[16]:
```python
def expEmin(N,sigma=0.1,d=8):
    return sigma**2*(1-(d+1)/N)

print(f'For N=45, we get an expected value for E_min of {expEmin(45):.3f}')
```

```
For N=45, we get an expected value for E_min of 0.008
```

Problem 2

Answer: [d] $\tilde{\omega}_1 < 0, ; \tilde{\omega}_2 > 0$

Derivation:

A hyperbola is the set of points in a plane whose distances from two fixed points, called foci, has an absolute difference that is equal to a positive constant. In formulae:

$$f(x_1, x_2) = x_1^2 - x_2^2 - r^2 = 0 , \tag{3}$$

where we assumed that the center is the origin of the coordinates $(0, 0)$, the hyperbola is equilateral, i.e. the asymptotes have unitary slopes, and $r \in \mathbb{R}$. The general case can be addressed by shifting the origin and/or rescaling the coordinates.

In the $\mathcal{Z}$ space, the point are classified by the sign of the following function:

$$\text{sgn}\left(\tilde{\omega}_0\Phi(x_0) + \tilde{\omega}_1\Phi(x_1) + \tilde{\omega}_2\Phi(x_2)\right) = \text{sgn}\left(\tilde{\omega}_0 + \tilde{\omega}_1 x_1^2 + \tilde{\omega}_2 x_2^2\right) . \tag{4}$$

In $\mathcal{X}$ space, a generic sample (x_1, x_2) is labelled by computing $\text{sgn}(-f(x_1, x_2))$, as illustrated in the following plot. In order to agree with the decision boundary in Eq.(4), i.e.

$$\text{sgn}\left(\tilde{\omega}_0 + \tilde{\omega}_1 x_1^2 + \tilde{\omega}_2 x_2^2)\right) = \text{sgn}(r^2 - x_1^2 + x_2^2) \tag{5}$$

we impose the following sets of constraints on the weights $\tilde{\omega}$:

$$\tilde{\omega}_0 > 0,\ \tilde{\omega}_1 < 0,\ \tilde{\omega}_2 > 0\,. \tag{6}$$

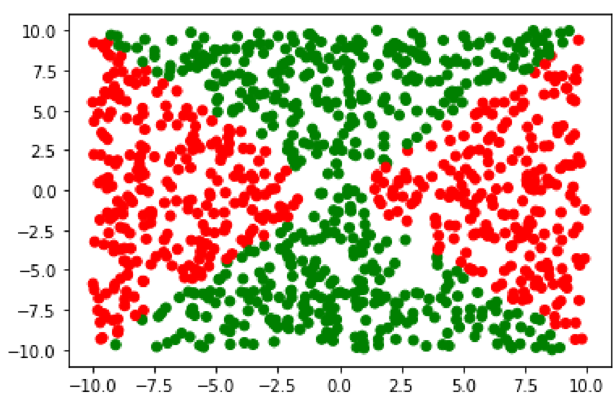
[26]:
```
import numpy as np
import matplotlib.pyplot as plt

def gen_uniform_points(N,d=2,vmin=[-1,-1],vmax=[1,1]):
    if(d!=len(vmin)|d!=len(vmax)):
        raise Exception('WARNING: Boundary values do not match the␣
 ↪dimensionality of the problem!')
    return np.random.uniform(low=vmin,high=vmax,size=(N,d))

def label_hyperbolic(pts, a=1,b=1,r=1):
    return [-np.sign(pts[i][0]*pts[i][0]/a**2-pts[i][1]*pts[i][1]/b**2-r**2) for␣
 ↪i in range(len(pts))]

def color_pts(label):
    #green is +1, red is -1
    col=[]
    for i in range(len(label)):
        if(label[i]>0): col.append('green')
        else: col.append('red')
    return col

N=1000
pts=gen_uniform_points(N,vmin=[-10,-10],vmax=[10,10])
label=label_hyperbolic(pts)
plt.scatter(pts[:,0],pts[:,1], color=color_pts(label))
plt.show()
```



Problem 3

Answer: [c] 15

Derivation:

Let's consider the general $\mathcal{Q}$th order polynomial transform $\Phi_{\mathcal{Q}}$ for the space $\S = \mathbb{R}^d$. We can find the dimensionality $\tilde{d}$ of the feature space $\mathcal{Z}$ by observing that we can form $C(d,k)$ different monomials of order k from the d initial coordinates, where

$$C(d,k) = \binom{d+k-1}{k}. \tag{7}$$

Since $\Phi_{\mathcal{Q}}$ will have all possible monomials up to order $\mathcal{Q}$ as transformed coordinates, the feature space $\mathcal{Z}$ will have a dimensionality

$$\tilde{d}(Q,d) = \sum_{k=1}^{Q} \binom{d+k-1}{k}. \tag{8}$$

For $d = 2$, we get

$$\tilde{d}(Q,2) = \sum_{k=1}^{Q} \binom{k+1}{k} = \sum_{k=1}^{Q} k+1 = \frac{Q(Q+3)}{2}. \tag{9}$$

The VC dimension of the set of the hypothesis in $\mathcal{Z}$ $d_{VC}(\mathcal{H}_\Phi)$ can be as high as the VC dimension of a linear model in the transformed space, in formulae:

$$d_{VC}(\mathcal{H}_\Phi) \leq \tilde{d} + 1. \tag{10}$$

For the case examined here, $Q = 4$, hence $d_{VC}(\mathcal{H}_\Phi) \leq \tilde{d}(4,2) + 1 = 15$.

Problem 4

Answer: [e] $2(ue^v - 2ve^{-u})(e^v + 2ve^{-u})$

Derivation:

$$\begin{aligned} \frac{\partial E(u,v)}{\partial u} &= 2(ue^v - 2ve^{-u})\frac{\partial}{\partial u}(ue^v - 2ve^{-u}) \\ &= 2(ue^v - 2ve^{-u})(e^v + 2ve^{-u}). \end{aligned} \tag{11}$$

Problems 5-6

Answers: [d] 10 **, [e]** $[0.045, 0.024]$

Code:

```
[65]: import math as m

def E(w):
    u=w[0]
    v=w[1]
    return np.double((u*m.e**v - 2*v*m.e**(-u))**2)

def gradE(w):
    u=w[0]
    v=w[1]
    duE=np.double(2*(u*m.e**v-2*v*m.e**(-u))*(m.e**v+2*v*m.e**(-u)))
    dvE=np.double(2*(-2*m.e**(-u)+m.e**v*u)*(m.e**v*u - 2*m.e**(-u)*v))
    return np.array([duE,dvE])

def grad_step(w,grad,eta):
    return w-eta*grad(w)

def grad_desc(E,gradE,Emin,init,eta=0.1,print_ans=True):
    w=init
    Niter=0
    while(E(w)>Emin):
        w=grad_step(w,gradE,eta)
        Niter=Niter+1
    if(print_ans==True):
        print(f'After {Niter} iterations, we found a minimum of the function at␣
 ↪[{w[0]:.3f},{w[1]:.3f}] with error value of {E(w):.2e}')
    return w,Niter

eps=np.double(10**(-14))
startpt=np.array([1,1])
w,Niter=grad_desc(E,gradE,eps,startpt)
```

```
After 10 iterations, we found a minimum of the function at [0.045,0.024] with
error value of 1.21e-15
```

Problem 7

Answer: [a] 10^{-1}

Code:

```
[74]: def grad_step_coord(w,gradE,eta):
          ustep=w[0]-eta*gradE(w)[0]
          w=np.array([ustep,w[1]],dtype=np.double)
          vstep=w[1]-eta*gradE(w)[1]
          return np.array([ustep,vstep],dtype=np.double)

      def grad_desc_coord(E,gradE,max_ite,init,eta=0.1,print_ans=True):
          w=init
          Niter=0
          while(Niter<max_ite):
              w=grad_step_coord(w,gradE,eta)
              Niter=Niter+1
          if(print_ans==True):
              print(f'After {Niter} iterations, we found a minimum of the function at␣
       ↪[{w[0]:.3f},{w[1]:.3f}] with error value of {E(w):.2e}')
          return w,Niter

      w,Niter=grad_desc_coord(E,gradE,15,startpt)
```

```
After 15 iterations, we found a minimum of the function at [6.297,-2.852] with
error value of 1.40e-01
```

Problems 8-9

Answers: [d] 0.100, **[a]** 350

Code:

```
[59]: import numpy as np
      import matplotlib.pyplot as plt
      import math as m

      def gen_uniform_points(N,d=2,vmin=[-1,-1],vmax=[1,1]):
          if(d!=len(vmin)|d!=len(vmax)):
              raise Exception('WARNING: Boundary values do not match the␣
       ↪dimensionality of the problem!')
          pts=np.random.uniform(low=vmin,high=vmax,size=(N,d))
          return  np.concatenate((np.ones(N)[:, np.newaxis], pts), axis=1)

      def gen_line():
          pts=gen_uniform_points(2)
          x1,x2=pts[0][1],pts[1][1]
```

```
    y1,y2=pts[0][2],pts[1][2]
    m=(y2-y1)/(x2-x1)
    b=(y1*x2-y2*x1)/(x2-x1)
    return m,b

def line(x,m,b):
    return x*m+b

def label_linear(pts,m,b):
    return np.array([np.sign(pts[i][2]-(m*pts[i][1]+b)) for i in␣
 ↪range(len(pts))])

def color_pts(label):
    #green is +1, red is -1
    col=[]
    for i in range(len(label)):
        if(label[i]>0): col.append('green')
        else: col.append('red')
    return col

def SGD_step(pt,label,w,eta):
    return w-eta*(-label*pt/(1+m.e**(label*np.dot(w,pt))))

def SGD_epoch(pts,label,w,eta):
    #set epoch random indices
    randindex=np.random.choice(range(len(label)),len(label), replace=False)
    for i in range(len(label)):
        w=SGD_step(pts[i],label[i],w,eta)
    return w

def cross_entropy(pts,label,w):
    Npts=len(label)
    Ein=0
    for i in range(Npts):
        Ein+=m.log(1+m.e**(-label[i]*np.dot(w,pts[i])))
    return Ein/Npts

def Eout_estimate(m,b,w,Nval=1000):
    pts=gen_uniform_points(Nval)
    true=label_linear(pts,m,b)
    return cross_entropy(pts,true,w)

def LogRegr_SGD(Npts,eta,stop,Nval=1000,plot=True):

    #generate data
    pts=gen_uniform_points(Npts)
    m,b=gen_line()
```

```python
    label=label_linear(pts,m,b)

    #initialization of SGD
    epoch=0
    w=np.zeros(3)

    #SGD epochs
    while True:
        wtemp=w
        w=SGD_epoch(pts,label,w,eta)
        dw=wtemp-w
        epoch+=1
        if(np.sqrt(np.dot(dw,dw))<stop): break

    #Eout estimate
    Eout=Eout_estimate(m,b,w,Nval=Nval)

    #plots
    if(plot==True):
        xaxis=np.linspace(-1,1,100)
        col=color_pts(label)
        plt.scatter(pts[:,1],pts[:,2],color=col)
        plt.plot(xaxis,line(xaxis,m,b),label='True')
        plt.plot(xaxis,line(xaxis,-w[1]/w[2],-w[0]/
 ↪w[2]),color='blue',linestyle='dashed',label='grad_desc')
        plt.xlim([-1, 1])
        plt.ylim([-1, 1])
        plt.legend()
        plt.show()

    return w, epoch, Eout
```

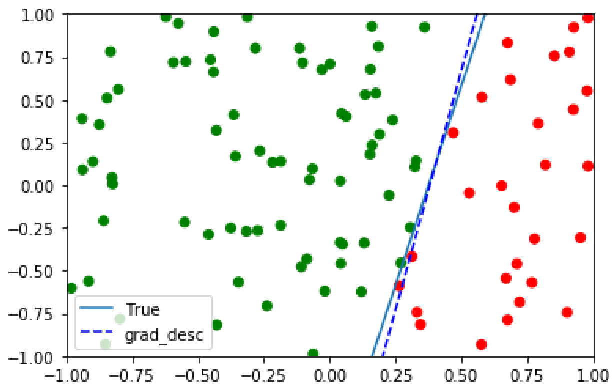
[69]:
```python
Nruns=100
Npts=100
eta=0.01
stop=0.01
Eavg=0
epochs=0

for i in range(Nruns):
    w,ep,Eout = LogRegr_SGD(Npts,eta,stop,plot=False)
    Eavg+=Eout
    epochs+=ep
```

```
print(f'For {Nruns} runs of Logistic Regression with SGD, with {Npts} points␣
↪each run and a learning rate of {eta}, we get the following average results:␣
↪\n Average Eout (cross-entropy error)={Eavg/Nruns:.3f} \n Average epoch of␣
↪convergence (with dwstop={stop})={epochs/Nruns:.0f}')
```

```
For 100 runs of Logistic Regression with SGD, with 100 points each run and a
learning rate of 0.01, we get the following average results:
 Average Eout (cross-entropy error)=0.104
 Average epoch of convergence (with dwstop=0.01)=349
```

[75]:
```
#plotted example
ex=LogRegr_SGD(Npts,eta,stop)
```



Problem 10

Answers: [e] $e_n(\mathbf{w}) = -\min(0, y_n \mathbf{w}^T \mathbf{x}_n)$

Derivation:

In Stochastic Gradient Descent (SGD) method, the weights are updated by picking one random point of the sample and computing the gradient of the error function $e_n(\mathbf{w})$. In formulae:

$$\mathbf{w} \to \mathbf{w} - \eta \nabla e_n(\mathbf{w}) \,, \tag{12}$$

where η is the learning rate.

For the error function proposed [e], the gradient is:

$$\nabla e_n(\mathbf{w}) = -\nabla \left(\min(0, y_n \mathbf{w}^T \mathbf{x}_n) \right) = \begin{cases} -y_n \mathbf{x}_n & \text{for } y_n \mathbf{w}^T \mathbf{x}_n < 0 \\ 0 & \text{for } y_n \mathbf{w}^T \mathbf{x}_n \geq 0 \end{cases} \tag{13}$$

We observe that $y_n \mathbf{w}^T \mathbf{x}_n < 0$ if and only if the point $\mathbf{x}_n$ is misclassified by the current weights.

In other words, if the point $\mathbf{x}_n$ is misclassified, the SGD algorithm with the error function defined in [e] and a learning rate of $\eta = 1$ will update the weights as the following:

$$\mathbf{w} \to \mathbf{w} + y_n \mathbf{x}_n \,. \tag{14}$$

This reproduces exactly the Perceptron Linear Algorithm (PLA) step.

[ ]:

```

```