CS 156a - Problem Set 7

Samuel Patrone, 2140749

November 14, 2022

The following notebook is publicly available here.

Contents

Problems 1-5

Answers: [d] $k = 6$**, [e]** $k = 7$**, [d]** $k = 6$**, [d]** $k = 6$**, [b]** $[0.1, 0.2]$

Code:

[50]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#import data

in_data=pd.read_csv('in.dta',header=None,delim_whitespace=True)
out_data=pd.read_csv('out.dta',header=None,delim_whitespace=True)

in_pts=in_data[[0, 1]].to_numpy()
in_y=in_data[2].to_numpy()

out_pts=out_data[[0, 1]].to_numpy()
out_y=out_data[2].to_numpy()
```

```
#non-linear transformation

def transform(pts,k):
    if(k>7):
        raise Exception('WARNING: k must be smaller than 7!')
    res=[]
    for i in range(len(pts)):
        x1=pts[i][0]
        x2=pts[i][1]
        res_temp=[]
        for k in range(k+1):
            res_full=[1,x1,x2,x1**2,x2**2,x1*x2,np.abs(x1-x2),np.abs(x1+x2)]
            res_temp.append(res_full[k])
        res.append(res_temp)
    return np.array(res)

def lin_reg(pts,y):
    return np.dot(np.linalg.pinv(pts),y)

def h(pts,w):
    return np.sign(np.dot(w,pts.T))

def validation(w,val_pts,val_y):
    gout=h(val_pts,w)
    testgout=(gout==val_y)
    return len(np.where(testgout==False)[0])/len(val_y)
```

[60]:
```
#Problem 1-2-5
print('Results for 25:10 (training:validation)\n')

train=in_pts[0:-10]
train_y=in_y[0:-10]
val=in_pts[25:]
val_y=in_y[25:]

k=[3,4,5,6,7]

for i in k:
    train_transf=transform(train,i)
    val_in=transform(val,i)
    val_out=transform(out_pts,i)
    w=lin_reg(train_transf,train_y)
    Eval_in=validation(w,val_in,val_y)
    Eval_out=validation(w,val_out,out_y)
    print(f'k={i} | Eval_in: {Eval_in:.2f} | Eval_out: {Eval_out:.2f}')
```

```
Results for 25:10 (training:validation)
```

```
k=3 | Eval_in: 0.30 | Eval_out: 0.42
k=4 | Eval_in: 0.50 | Eval_out: 0.42
k=5 | Eval_in: 0.20 | Eval_out: 0.19
k=6 | Eval_in: 0.00 | Eval_out: 0.08
k=7 | Eval_in: 0.10 | Eval_out: 0.07
```

[59]:
```python
#Problem 3-4-5
print('Results for 10:25 (training:validation)\n')

val=in_pts[0:-10]
val_y=in_y[0:-10]
train=in_pts[25:]
train_y=in_y[25:]

k=[3,4,5,6,7]

for i in k:
    train_transf=transform(train,i)
    val_in=transform(val,i)
    val_out=transform(out_pts,i)
    w=lin_reg(train_transf,train_y)
    Eval_in=validation(w,val_in,val_y)
    Eval_out=validation(w,val_out,out_y)
    print(f'k={i} | Eval_in: {Eval_in:.2f} | Eval_out: {Eval_out:.2f}')
```

```
Results for 10:25 (training:validation)

k=3 | Eval_in: 0.28 | Eval_out: 0.40
k=4 | Eval_in: 0.36 | Eval_out: 0.39
k=5 | Eval_in: 0.20 | Eval_out: 0.28
k=6 | Eval_in: 0.08 | Eval_out: 0.19
k=7 | Eval_in: 0.12 | Eval_out: 0.20
```

Problem 6

Answer: [d] $[0.5, 0.5, 0.4]$

Code:

[93]:
```python
e1_avg=0
e2_avg=0
min_avg=0

Nruns=1000000

for i in range(Nruns):
    e1,e2=np.random.rand(),np.random.rand()
    e1_avg+=e1
```

```
    e2_avg+=e2
    min_avg+=min(e1,e2)

print(f'After Nruns={Nruns:.1e}, we find the following average values:
 ↪\n\n[e1,e2,min(e1,e2)]=[{e1_avg/Nruns:.1f},{e2_avg/Nruns:.1f},{min_avg/Nruns:.
 ↪2f}]')
```

```
After Nruns=1.0e+06, we find the following average values:

[e1,e2,min(e1,e2)]=[0.5,0.5,0.33]
```

Problem 7

Answer: [c] $\sqrt{9+4\sqrt{6}}$

Derivation:

In the following, we express the squared error as an analytic function of the variable ρ using the leave-one-out algorithm with the cross-validation error.

Given two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, the best fit for the hypothesis set $h_0(x) = b$ is given by the constant line passing in between them, i.e.

$$b = \frac{y_1 + y_2}{2}. \tag{1}$$

For the hypothesis set $h_1(x) = ax + c$, the best fit is given instead by:

$$a = \frac{y_2 - y_1}{x_2 - x_1} b = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1} \tag{2}$$

Hence, for the points $p_1 = (-1, 0)$, $p_2 = (\rho, 1)$ and $p_3 = (1, 0)$, using the leave-one-out algorithm, we have the following fits:

$$p_1, p_2 \to b = \frac{1}{2} \quad (a, c) = \left(\frac{1}{1+\rho}, \frac{1}{1+\rho}\right) \tag{3}$$

$$p_1, p_3 \to b = 0 \quad (a, c) = (0, 0) \tag{4}$$

$$p_2, p_3 \to b = \frac{1}{2} \quad (a, c) = \left(\frac{1}{1-\rho}, \frac{-1}{1-\rho}\right) \tag{5}$$

The average square error on the left-out-point is therefore:

$$E_0 = \frac{1}{2} E_1 = \frac{1}{3}\left[1 + \left(\frac{4}{1+\rho}\right)^2 + \left(\frac{4}{1-\rho}\right)^2\right]. \tag{6}$$

Solving for ρ the equation $E_0 = E_1$, we find

$$\rho = \sqrt{9 + 4\sqrt{6}}. \tag{7}$$

Problems 8-10

Answers: **[c]** 60%, **[d]** 65%, **[b]** 3

Code:

[92]:
```
import numpy as np
import matplotlib.pyplot as plt
import math as m
from sklearn import svm

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 h(w,data):
    return np.sign(np.dot(w,data.T))

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 Eout(m,b,w,Nval=1000):
```

```
    pts=gen_uniform_points(Nval)
    true=label_linear(pts,m,b)
    g=h(w,pts)
    testg=(g==true)
    Eout=len(np.where(testg==False)[0])
    return Eout/Nval

def PLA(pts,y,plot=False):

    #initialization before learning
    w=np.zeros(3)
    converged=False
    iterations=0

    #learning algorithm
    while(converged==False):
        g=h(w,pts)
        testg=(g==y)
        misclassified=np.where(testg==False)[0]
        if(len(misclassified)>0):
            j=np.random.randint(len(misclassified))
            i=misclassified[j]
            w+=y[i]*pts[i]
            iterations+=1
        g=h(w,pts)
        converged=np.all(g==y)

    if(plot==True):
        x=np.linspace(-1,1,100)
        col=color_pts(y)
        plt.scatter(pts[:,1],pts[:,2],color=col)
        plt.plot(x,line(x,-w[1]/w[2],-w[0]/w[2]),color='blue',linestyle='dashed')
        plt.xlim([-1, 1])
        plt.ylim([-1, 1])
    return w, iterations

def SVM(pts,y,plot=False):
    clf = svm.SVC(C=np.Inf, kernel='linear', cache_size=20000)
    clf.fit(pts[:,1:3], y)
    w=clf.coef_[0]
    w=np.append(clf.intercept_[0],w)
    if(plot==True):
        x=np.linspace(-1,1,100)
        plt.plot(x,line(x,-w[1]/w[2],-w[0]/w[2]),color='blue')
    return w,sum(clf.n_support_)
```

[95]:
```
Npts=10
Nruns=1000

out_perf=0
for i in range(Nruns):
    check=True
    while(check==True):
        pts=gen_uniform_points(Npts)
        m,b=gen_line()
        y=label_linear(pts,m,b)
        check=(np.abs(np.sum(y))==Npts)
    wp,it=PLA(pts,y)
    ws,vec=SVM(pts,y)
    if(Eout(m,b,wp)>Eout(m,b,ws)): out_perf+=1

print(f"With Npts={Npts}, averaged over Nruns={Nruns}, SVM performs better than␣
 ↪PLA {out_perf/10:.0f}% of the times!")
```

```
With Npts=10, averaged over Nruns=1000, SVM performs better than PLA 60% of the
times!
```

[100]:
```
Npts=100
Nruns=1000

out_perf=0
vec=0
for i in range(Nruns):
    check=True
    while(check==True):
        pts=gen_uniform_points(Npts)
        m,b=gen_line()
        y=label_linear(pts,m,b)
        check=(np.abs(np.sum(y))==Npts)
    wp,it=PLA(pts,y)
    ws,vec_run=SVM(pts,y)
    vec+=vec_run
    if(Eout(m,b,wp)>Eout(m,b,ws)): out_perf+=1

print(f"With Npts={Npts}, averaged over Nruns={Nruns}, SVM performs better than␣
 ↪PLA {out_perf/10:.0f}% of the times!\nThe average number of support vectors is␣
 ↪{vec/Nruns:.0f}")
```

```
With Npts=100, averaged over Nruns=1000, SVM performs better than PLA 61% of the
times!
The average number of support vectors is 3
```

[ ]:
```

```