```python
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 19 01:21:35 2021

@author: 11327
"""

import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cvx
import scipy

# Ex 3
# b
# read data from txt file
xclass0 = np.matrix(np.loadtxt('./data/homework4_class0.txt'))
xclass1 = np.matrix(np.loadtxt('./data/homework4_class1.txt'))

# create x
x = np.concatenate((xclass0,xclass1),axis=0)
[rowx,colx] = np.shape(x)
x = np.concatenate((x,np.ones((rowx,1))),axis=1)

# create y
[rowx0,colx0] = np.shape(xclass0)
[rowx1,colx1] = np.shape(xclass1)
y0 = np.zeros((rowx0,1))
y1 = np.ones((rowx1,1))
y = np.concatenate((y0,y1),axis=0)

# CVX
lambd = 0.0001
N = rowx0 + rowx1
theta       = cvx.Variable((3,1))
loss        = - cvx.sum(cvx.multiply(y, x @ theta)) \
              + cvx.sum(cvx.log_sum_exp( cvx.hstack([np.zeros((N,1)), x @ theta]), axis=1 ) )
reg         = cvx.sum_squares(theta)
prob        = cvx.Problem(cvx.Minimize(loss/N + lambd*reg))
prob.solve()
w = theta.value

# c
# calculate the boundary
xb = np.linspace(-4,8,100)
yb = (-w[0]*xb-w[2])/w[1]

# do the plot
'''
plt.figure()
plt.scatter(xclass0[:,0].tolist(),xclass0[:,1].tolist())
plt.scatter(xclass1[:,0].tolist(),xclass1[:,1].tolist(), c='g')

plt.plot(xb,yb,c='b')
plt.show()
'''
# d
# create testing sites
n = 100
testing = np.linspace(-5,10,n)
# y = np.linspace(-5,10,n)
xv,yv = np.meshgrid(testing,testing)
```

```
boundary = np.zeros((n,n))

# find parameters
miu0 = np.zeros(colx0)
for i in range(colx0):
    miu0[i] = np.mean(xclass0[i])

miu1 = np.zeros(colx1)
for i in range(colx1):
    miu1[i] = np.mean(xclass1[i])

Sigma0 = np.cov(xclass0.T)
Sigma1 = np.cov(xclass1.T)

d = rowx0
abs_Sigma1 = np.linalg.det(Sigma1)
abs_Sigma0 = np.linalg.det(Sigma0)
inv_Sigma1 = np.linalg.inv(Sigma1)
inv_Sigma0 = np.linalg.inv(Sigma0)
const = np.power((2*np.pi),d)

# do Bayesian Decision
for i in range(100):
    for j in range(100):
        block = np.matrix([testing[i],testing[j]]).T
        # block = np.matrix((x[i,0],x[i,1])).T
        c1 = 1/(np.sqrt(const*abs_Sigma1)) * np.exp(-0.5*np.dot(np.dot((block-miu1).T,inv_Sigma1
        c0 = 1/(np.sqrt(const*abs_Sigma0)) * np.exp(-0.5*np.dot(np.dot((block-miu0).T,inv_Sigma0
        if c1[0,0] > c0[0,0]:
            boundary[i,j] = 1
        elif c1[0,0] < c0[0,0]:
            boundary[i,j] = 0
'''
plt.contour(testing,testing,boundary)
plt.show()
'''
# Ex 4
# a
m,n = 100,100
K = np.zeros((m,n))

h = 1
x = x[:,0:2]
for i in range(m):
    for j in range(n):
        K[i,j] = np.exp(-np.power(np.linalg.norm(x[i,:]-x[j,:],ord=1),2)/h)

# print(K[47:52,47:52])

# c
lambd = 0.001
alpha     = cvx.Variable((N,1))
loss        = - cvx.sum(cvx.multiply(y, K @ alpha)) \
              + cvx.sum(cvx.log_sum_exp( cvx.hstack([np.zeros((N,1)), K @ alpha]), axis=1 ) )
reg         = cvx.sum(cvx.quad_form(alpha, K))
prob        = cvx.Problem(cvx.Minimize(loss/N + lambd*reg))
prob.solve()
ALPHA = alpha.value
```