

```

# -*- coding: utf-8 -*-
"""
Created on Thu Mar  4 23:17:38 2021

@author: 11327
"""

import numpy as np
import matplotlib.pyplot as plt
import cv2
from PIL import Image
import cvxpy as cvx

train_cat = np.matrix(np.loadtxt('train_cat.txt', delimiter = ','))
train_grass = np.matrix(np.loadtxt('train_grass.txt', delimiter = ','))

Y = plt.imread('cat_grass.jpg') / 255
# Y = cv2.imread('rabbit.jpg',0) / 255

# EX2
# 2
# i
row0,col0 = np.shape(train_grass)
miu0 = np.zeros(row0)
for i in range(row0):
    miu0[i] = np.mean(train_grass[i])

row1,col1 = np.shape(train_cat)
miu1 = np.zeros(row1)
for i in range(row1):
    miu1[i] = np.mean(train_cat[i])

# ii
Sigma0 = np.cov(train_grass)
Sigma1 = np.cov(train_cat)

# iii
K0 = col0
K1 = col1
pi0 = K0 / (K1+K0)
pi1 = K1 / (K1+K0)

# c
M,N = Y.shape
prediction = np.zeros((M,N))
'''
for i in range(M-8):
    for j in range(N-8):
        block = Y[i:i+8, j:j+8] # This is a 8x8 block
        block = np.reshape(block,(64,1))
        C1 = (-1/2)*np.dot(np.dot((block-miu1).T,np.linalg.inv(Sigma1)),(block-miu1))+np.log(pi1)
        C0 = (-1/2)*np.dot(np.dot((block-miu0).T,np.linalg.inv(Sigma0)),(block-miu0))+np.log(pi0)
        if C1[0,0] > C0[0,0]:
            prediction[i,j] = 1
        elif C1[0,0] < C0[0,0]:
            prediction[i,j] = 0

prediction = prediction*255
im = Image.fromarray(prediction)
im = im.convert('L')
im.save('outfile.png')

```

```

# d
prediction = prediction / 255

rowp,colp = prediction.shape
truth = plt.imread('truth.png')
MAE = 0
for i in range(rowp-8):
    for j in range(colp-8):
        MAE = MAE + np.abs(prediction[i,j]-truth[i,j])

MAE = 1/(rowp*colp) * MAE
'''

# Ex 3
# b
truth = plt.imread('truth.png')
# Number of true positive in ground truth
TP_total = np.count_nonzero(truth)
# Number of true negative in ground truth
TN_total = len(truth.ravel()) - TP_total
'''

d = 64
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)
div = np.zeros((M,N))

for i in range(M-8):
    for j in range(N-8):
        block = Y[i:i+8, j:j+8] # This is a 8x8 block
        block = np.reshape(block,(64,1))
        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)
        div[i,j] = np.log(c1[0,0]/c0[0,0])

n_tau = 101
PF = np.zeros(n_tau)
PD = np.zeros(n_tau)
tauset = np.linspace(-200,100,n_tau)
for k in range(n_tau):
    tau = (tauset[k])
    prediction = np.zeros(Y.shape)
    true_positive = 0; true_negative = 0
    false_positive = 0; false_negative = 0
    prediction = div > tau
    for i in range(M - 8):
        for j in range(N - 8):
            if (prediction[i][j]==1) and (truth[i][j]>=0.5): true_positive += 1
            if (prediction[i][j]==1) and (truth[i][j]<=0.5): false_positive += 1
            if (prediction[i][j]==0) and (truth[i][j]>=0.5): false_negative += 1
            if (prediction[i][j]==0) and (truth[i][j]<=0.5): true_negative += 1

# Calculate Pd and Pf

PD[k] = true_positive/TP_total
PF[k] = false_positive/TN_total

plt.figure()

```

```

plt.plot(PF,PD)
plt.xlabel('PF')
plt.ylabel('PD')
plt.title('ROC curve')

tau_op = pi0 / pi1
tau = tau_op
prediction = np.zeros(Y.shape)
true_positive = 0; true_negative = 0
false_positive = 0; false_negative = 0
prediction = div > tau
for i in range(M - 8):
    for j in range(N - 8):
        if (prediction[i][j]==1) and (truth[i][j]>=0.5): true_positive += 1
        if (prediction[i][j]==1) and (truth[i][j]<=0.5): false_positive += 1
        if (prediction[i][j]==0) and (truth[i][j]>=0.5): false_negative += 1
        if (prediction[i][j]==0) and (truth[i][j]<=0.5): true_negative += 1

PD_op = true_positive/TP_total
PF_op = false_positive/TN_total

plt.plot(PF_op,PD_op, 'ro', markersize=10)
plt.imshow()
'''

# d
d = 64
A = np.vstack([train_cat.T,train_grass.T])
b = np.vstack([np.ones((K1,1)), -1*np.ones((K0,1))])
theta = cvx.Variable((d,1))
objective = cvx.Minimize(cvx.sum_squares(A@theta - b))
prob = cvx.Problem(objective)
prob.solve()
sol = theta.value

pro = np.zeros((M,N))
for i in range(M-8):
    for j in range(N-8):
        block = Y[i:i+8, j:j+8] # This is a 8x8 block
        block = np.reshape(block,(64,1))
        pro[i,j] = (np.dot(sol.T,block))

n_tau = 100
PF = np.zeros(n_tau)
PD = np.zeros(n_tau)
tauset = np.linspace(-1.2,-0.1,n_tau)
for k in range(n_tau):
    tau = (tauset[k])
    prediction = np.zeros(Y.shape)
    true_positive = 0; true_negative = 0
    false_positive = 0; false_negative = 0
    prediction = pro > tau
    for i in range(M - 8):
        for j in range(N - 8):
            if (prediction[i][j]==1) and (truth[i][j]>=0.5): true_positive += 1
            if (prediction[i][j]==1) and (truth[i][j]<=0.5): false_positive += 1
            if (prediction[i][j]==0) and (truth[i][j]>=0.5): false_negative += 1
            if (prediction[i][j]==0) and (truth[i][j]<=0.5): true_negative += 1

# Calculate Pd and Pf

```

```
PD[k] = true_positive/TP_total
PF[k] = false_positive/TN_total

plt.figure()
plt.plot(PF,PD)
plt.xlabel('PF')
plt.ylabel('PD')
plt.title('ROC curve')
```