Lab1

August 15, 2019

1 Lab1: Color Image Segmentation Using EM algorithm (100 points).

1.1 Some ground rules and instructions.

1.1.1 Lab1 is due by Friday August 23, 9 PM, and is expected to require 4–6 hours of work depending on your Python coding skills (excluding the time to set up the environment with libraries installed).

1.1.2 Hints are provided where necessary, but you are strongly encouraged to start early. Late submissions will be exponentially penalized (20% for 1 second delay, 50% for 48 hour delay and so on).

1.1.3 You are welcome to discuss with friend or refer to resources online for solving the question, but plagiarism will be strictly dealt with and penalized as discussed in Lecture 1.

1.1.4 You can also utilize the help of TA's during the Monday and Thursday lab sessions. Note that TA's will only guide you, do not expect them to PROVIDE THE SOLUTION.

1.1.5 This exercise should familiarize you with the image processing and matrix manipulation commands available as part of Python.

1.1.6 Excellent solutions are entitled to extra points. Good luck!

[ ]:
```
#Modules to install
import matplotlib.pyplot as plt
import os
from os.path import join
import numpy as np
from PIL import Image
import matplotlib.image as mpimg
from skimage.color import rgb2gray
from skimage.color import label2rgb
from skimage.filters import gaussian
from sklearn.cluster import KMeans
```

[ ]:
```
# to clear workspace and the display
plt.close('all')
clear = lambda: os.system('clear')
clear()
```

[ ]:
```
np.random.seed(110) #for reproducability of results
imgNames = ['water_coins','jump','tiger']#{'balloons', 'mountains', 'nature',␣
 ↪'ocean', 'polarlights'};
segmentCounts = [2,3,4,5]

for imgName in imgNames:
    for SegCount in segmentCounts:
        # Load the imageusing OpenCV
        img = """ Read Image using mplib library-- 2 points """
        print('Using Matplotlib Image Library: Image is of datatype ',img.
 ↪dtype,'and size ',img.shape) # Image is of type float

        # Load the Pillow-- the Python Imaging Library
        img = """ Read Image using PILLOW-- 3 points"""
        print('Using Pillow (Python Image Library): Image is of datatype ',img.
 ↪dtype,'and size ',img.shape) # Image is of type uint8

        #%% %Define Parameters
        nSegments = SegCount   # of color clusters in image
        nPixels = """ Compute number of image pixels from image dimensions-- 2␣
 ↪points""";    # Image can be represented by a matrix of size nPixels*nColors
        maxIterations = 20; #maximum number of iterations allowed for EM␣
 ↪algorithm.
        nColors = 3;
        #%% Determine the output path for writing images to files
        outputPath = join(''.join(['Output/',str(SegCount), '_segments/',␣
 ↪imgName , '/']));
        if not(os.path.exists(outputPath)):
            os.makedirs(outputPath)
        """ save input image as *0.png* under outputPath-- 3 points""" #save␣
 ↪using Matplotlib image library

        pixels = img
        pixels = """ Reshape pixels as a nPixels X nColors X 1 matrix-- 5␣
 ↪points"""

        #%%
        """ Initialize pi (mixture proportion) vector and mu matrix (containing␣
 ↪means of each distribution)
            Vector of probabilities for segments... 1 value for each segment.
            Best to think of it like this...
            When the image was generated, color was determined for each pixel␣
 ↪by selecting
            a value from one of "n" normal distributions. Each value in this␣
 ↪vector
```

```
            corresponds to the probability that a given normal distribution was␣
↪chosen."""


        """ Initial guess for pi's is 1/nSegments. Small amount of noise added␣
↪to slightly perturb
            GMM coefficients from the initial guess"""

        pi = 1/nSegments*(np.ones((nSegments, 1),dtype='float'))
        increment = np.random.normal(0,.0001,1)
        for seg_ctr in range(len(pi)):
            if(seg_ctr%2==1):
                pi[seg_ctr] = pi[seg_ctr] + increment
            else:
                pi[seg_ctr] = pi[seg_ctr] - increment

         #%%
        """Similarly, the initial guess for the segment color means would be a␣
↪perturbed version of [mu_R, mu_G, mu_B],
            where mu_R, mu_G, mu_B respectively denote the means of the R,G,B␣
↪color channels in the image.
            mu is a nSegments X nColors matrrix,(seglabels*255).np.asarray(int)␣
↪where each matrix row denotes mean RGB color for a particcular segment"""

        mu = """Initialize mu to 1/nSegments*['ones' matrix (whose elements are␣
↪all 1) of size nSegments X nColors] -- 5 points"""  #for even start
        #add noise to the initialization (but keep it unit)
        for seg_ctr in range(nSegments):
            if(seg_ctr%2==1):
                increment = np.random.normal(0,.0001,1)
            for col_ctr in range(nColors):
                 if(seg_ctr%2==1):
                    mu[seg_ctr,col_ctr] = np.mean(pixels[:,col_ctr]) + increment
                 else:
                    mu[seg_ctr,col_ctr] = np.mean(pixels[:,col_ctr]) - increment

        #%% EM-iterations begin here. Start with the initial (pi, mu) guesses  ␣
↪

        mu_last_iter = mu;
        pi_last_iter = pi;


        for iteration in range(maxIterations):
            """%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
            % -----------------   E-step  -----estimating likelihoods and␣
↪membership weights (Ws)
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"""

        print(''.join(['Image: ',imgName,' nSegments: ',str(nSegments),'␣
↪iteration: ',str(iteration+1), ' E-step']))
        # Weights that describe the likelihood that pixel denoted by␣
↪"pix_import scipy.miscctr" belongs to a color cluster "seg_ctr"
        Ws = np.ones((nPixels,nSegments),dtype='float')  # temporarily␣
↪reinitialize all weights to 1, before they are recomputed

        """ logarithmic form of the E step."""

        for pix_ctr in range(nPixels):
            # Calculate Ajs
            logAjVec = np.zeros((nSegments,1),dtype='float')
            for seg_ctr in range(nSegments):
                x_minus_mu_T  = np.transpose(pixels[pix_ctr,:]-(mu[seg_ctr,:
↪])[np.newaxis].T)
                x_minus_mu    = ((pixels[pix_ctr,:]-(mu[seg_ctr,:])[np.
↪newaxis].T))
                logAjVec[seg_ctr] = np.log(pi[seg_ctr]) - .5*(np.
↪dot(x_minus_mu_T,x_minus_mu))

            # Note the max
            logAmax = max(logAjVec.tolist())

            # Calculate the third term from the final eqn in the above link
            thirdTerm = 0;
            for seg_ctr in range(nSegments):
                thirdTerm = thirdTerm + np.exp(logAjVec[seg_ctr]-logAmax)

            # Here Ws are the relative membership weights(p_i/sum(p_i)),␣
↪but computed in a round-about way
            for seg_ctr in range(nSegments):
                logY = logAjVec[seg_ctr] - logAmax - np.log(thirdTerm)
                Ws[pix_ctr][seg_ctr] = np.exp(logY)

        """%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % -----------------   M-step  --------------------
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"""

        print(''.join(['Image: ',imgName,' nSegments: ',str(nSegments),'␣
↪iteration: ',str(iteration+1), ' M-step: Mixture coefficients']))
        #% temporarily reinitialize mu and pi to 0, before they are␣
↪recomputed
```

```
            mu = np.zeros((nSegments,nColors),dtype='float') # mean color for␣
↪each segment
            pi = np.zeros((nSegments,1),dtype='float') #mixture coefficients


            for seg_ctr in range(nSegments):

                denominatorSum = 0;
                for pix_ctr in range(nPixels):
                    """Update RGB color vector of mu[seg_ctr] as current␣
↪mu[seg_ctr] + pixels[pix_ctr,:] times Ws[pix_ctr,seg_ctr] -- 5 points"""
                    denominatorSum = denominatorSum + Ws[pix_ctr][seg_ctr]

                """Compute mu[seg_ctr] and denominatorSum directly without the␣
↪'for loop'-- 10 points.
                    If you find the replacement instruction, comment out the for␣
↪loop with your solution"
                    Hint: Use functions squeeze, tile and reshape along with␣
↪sum"""
                ## Update mu
                mu[seg_ctr,:] =  mu[seg_ctr,:]/ denominatorSum;
                ## Update pi
                pi[seg_ctr] = denominatorSum / nPixels; #sum of weights (each␣
↪weight is a probability) for given segment/total num of pixels


            print(np.transpose(pi))

            muDiffSq = np.sum(np.multiply((mu - mu_last_iter),(mu -␣
↪mu_last_iter)))
            piDiffSq = np.sum(np.multiply((pi - pi_last_iter),(pi -␣
↪pi_last_iter)))

            if (muDiffSq < .0000001 and piDiffSq < .0000001): #sign of␣
↪convergence
                print('Convergence Criteria Met at Iteration: ',iteration, '--␣
↪Exiting code')
                break;


            mu_last_iter = mu;
            pi_last_iter = pi;


            ##Draw the segmented image using the mean of the color cluster as␣
↪the
```

```
            ## RGB value for all pixels in that cluster.
            segpixels = np.array(pixels)
            cluster = 0
            for pix_ctr in range(nPixels):
                cluster = np.where(Ws[pix_ctr,:] == max(Ws[pix_ctr,:]))
                vec     = np.squeeze(np.transpose(mu[cluster,:]))
                segpixels[pix_ctr,:] =  vec.reshape(vec.shape[0],1)

            """ Save segmented image at each iteration. For displaying␣
↪consistent image clusters, it would be useful to blur/smoothen the segpixels␣
↪image using a Gaussian filter.
                Prior to smoothing, convert segpixels to a Grayscale image, and␣
↪convert the grayscale image into clusters based on pixel intensities"""

            segpixels = np.reshape(segpixels,(img.shape[0],img.
↪shape[1],nColors)) ## reshape segpixels to obtain R,G, B image
            segpixels = """convert segpixels to uint8 gray scale image and␣
↪convert to grayscale-- 5 points""" #convert to grayscale
            kmeans = """ Use kmeans from sci-kit learn library to cluster␣
↪pixels in gray scale segpixels image to *nSegments* clusters-- 10 points"""
            seglabels = """ reshape kmeans.labels_ output by kmeans to have the␣
↪same size as segpixels -- 5 points"""
            seglabels = "Use np.clip, Gaussian smoothing with sigma =2 and␣
↪label2rgb functions to smoothen the seglabels image, and output a float RGB␣
↪image with pixel values between [0--1]-- 20 points"""
            mpimg.imsave(''.join([outputPath,str(iteration+1),'.
↪png']),seglabels) #save the segmented output

            """ Display the 20th iteration (or final output in case of␣
↪convergence) segmentation images with nSegments = 2,3,4,5 for the three␣
↪images-- this will be a 3 row X 4 column image matrix-- 15 points"""
            """ Comment on the results obtained, and discuss your understanding␣
↪of the Image Segmentation problem in general-- 10 points """
```