chettah-colour

May 13, 2019

1 K-Means clustering

```
1: Choose the number of K clusters
2. Select at random "K" points , the centroids
3. Assign each data points to the closest centroid , which  forms K clusters (Closests--> Eucli
4. Compute an dplace the new centroid of each other.
5. Reassign each data point to the new closest centroid. if any reassignmnet took place, go to
```

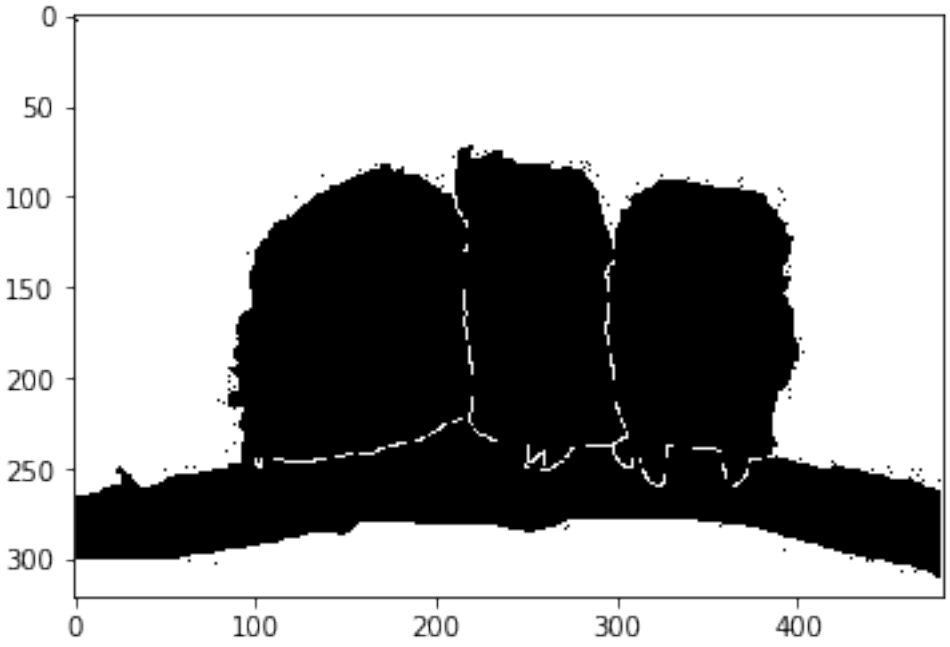
In [2]:
```
import math
from PIL import Image
from pylab import *
import matplotlib.cm as cm
import scipy as sp
import random
from collections import defaultdict
import operator

im = Image.open('F:\janani\Test_images\input2.jpg')
arr = np.asarray(im)

out = Image.open('F:\janani\Test_images\out2.jpg').convert('L')
arr_out = np.asarray(out)

rows,columns = np.shape(arr_out)
print('background pixel level',arr_out[0][0])

#print '\nrows',rows,'columns',columns
ground_out = np.zeros((rows,columns))


''' Converting ground truth image into binary image for evaluation'''

for i in range(rows):
    for j in range(columns):
        if arr_out[i][j] > 120:
            ground_out[i][j] = 0
```

```
        else:
            ground_out[i][j] = 1

plt.figure()
plt.imshow(ground_out, cmap="Greys_r")
plt.show()


shape = np.shape(arr)

rows = shape[0]
columns = shape[1]

'''obtaining 6 random centroid points'''

r_points = [ random.randint(0, 255) for i in range(6) ]
g_points = [ random.randint(0, 255) for i in range(6) ]
b_points = [ random.randint(0, 255) for i in range(6) ]

grey_l = defaultdict(list)

''' Grey scale levels corresponding to 6 clusters'''

grey1 = 40
grey2 = 80
grey3 = 120
grey4 = 160
grey5 = 200
grey6 = 240

grey_l[0] = 40
grey_l[1] = 80
grey_l[2] = 120
grey_l[3] = 160
grey_l[4] = 200
grey_l[5] = 240



g = defaultdict(list)

g2 = []
g3 = []
g4 = []
g5 = []
g6 = []

end = np.zeros((rows,columns))
```

```
zavg = [0,0,0]

''' computing average centroids after every iteration'''

def find_centroids(g):
    red_cent_list = []
    blue_cent_list = []
    green_cent_list = []
    #print '0 shape',np.shape(g)
    for i in range(0,6):
        array = np.matrix(g[i])
        avg = np.mean(array,0)
        #print '\naverage values',avg
        pavg = np.ravel(avg)
        #print '2 shape', np.shape(pavg)
        #print 'pavg', pavg
        if not len(pavg):
            red_cent_list.append(zavg[0])
            blue_cent_list.append(zavg[1])
            green_cent_list.append(zavg[2])
        else:
            red_cent_list.append(pavg[0])
            blue_cent_list.append(pavg[1])
            green_cent_list.append(pavg[2])
    return[red_cent_list,blue_cent_list,green_cent_list]


''' Computing 10 iterations to obtain converged centroids of six clusters'''

for it in range(0,10):
    print('\niteration',it)
    g= defaultdict(list)
    for r in range(rows):
        for c in range(columns):
            img = arr[r][c]
            #print '\nimg', img
            red = img[0]
            green = img[1]
            blue =  img[2]
            #print '\n red',red,'blue',blue,'green',green

            distance_list = []
            for k in range(0,6):
                ''' computing absolute distance from each of the cluster and assigning
                #print '\n red ref point',r_points[k],'blue ref point ',b_points[k],'g
                distance = math.sqrt(((int(r_points[k])- red)**2)+((int(g_points[k]) -
                #print '\ndistance',distance
                distance_list.append(distance)
```

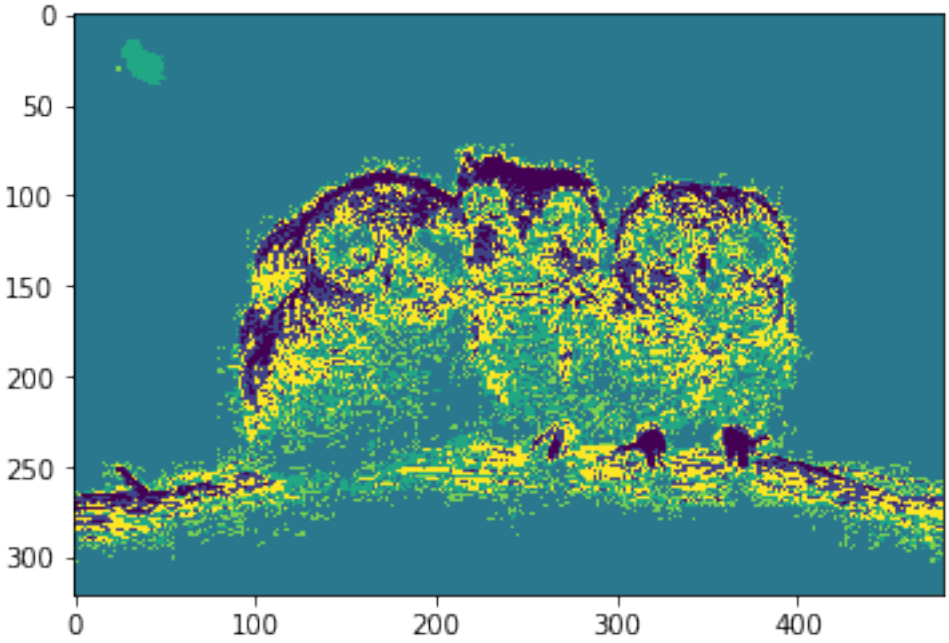
```
            #print '\ndistance list',distance_list
            index, value = min(enumerate(distance_list), key=operator.itemgetter(1))
            end[r][c] = grey_l[index]
            #print '\nindex',index
            g[index].append([red,blue,green])
    centroids= find_centroids(g)
    #print 'centroids',centroids
    r_points = []
    b_points = []
    g_points = []
    r_points = centroids[0]
    b_points = centroids[1]
    g_points = centroids[2]
    #print '\nr points',r_points
    #print '\nb points',b_points
    #print '\ng points',g_points


''' From observation we know that ground truth image pixel[0,0] is part of background.

result = np.zeros((rows,columns))
ref_val = end[0][0]
#print '\nref_val',ref_val
for i in range(rows):
    for j in range(columns):
        if end[i][j] ==  ref_val:
            result[i][j] = 1

        else:
            result[i][j] = 0


''' ********************************** Calculation of Tpr, Fpr, F-Score **************


tp = 0
tn = 0
fn = 0
fp = 0

for i in range(rows):
    for j in range(columns):
        if ground_out[i][j] == 1 and result[i][j] == 1:
            tp = tp + 1
        if ground_out[i][j] == 0 and result[i][j] == 0:
            tn = tn + 1
        if ground_out[i][j] == 1 and result[i][j] == 0:
            fn = fn + 1
```

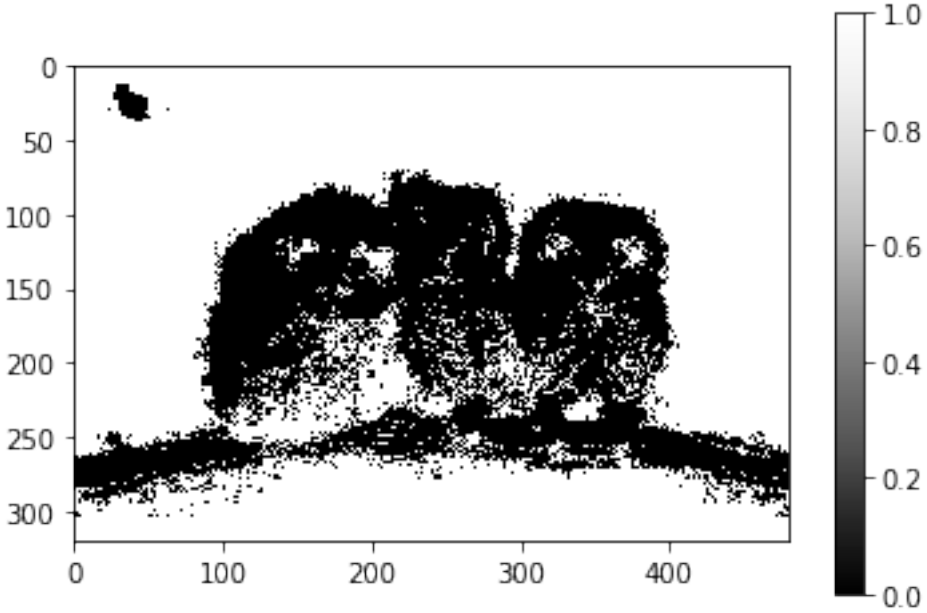
```
                if ground_out[i][j] == 0 and result[i][j] == 1:
                    fp = fp + 1



        print('\n************Calculation of Tpr, Fpr, F-Score*******************')

        #TP rate = TP/TP+FN
        tpr= float(tp)/(tp+fn)
        print("\nTPR is:",tpr)

        #fp rate is
        fpr= float(fp)/(fp+tn)
        print("\nFPR is:",fpr)

        #F-score as 2TP/(2TP + FP + FN)
        fscore = float(2*tp)/((2*tp)+fp+fn)
        print("\nFscore:",fscore)


        plt.figure()
        plt.imshow(end)
        plt.show()


        '''displaying the clusters in different gray scale levels'''


        plt.figure()
        plt.imshow(result, cmap="Greys_r")
        plt.colorbar()
        plt.show()

background pixel level 0
```




```
iteration 0

iteration 1

iteration 2

iteration 3

iteration 4

iteration 5

iteration 6

iteration 7

iteration 8

iteration 9

**********Calculation of Tpr, Fpr, F-Score**********

TPR is: 0.9759515723547832

FPR is: 0.27010312514715346
```

```
Fscore: 0.9012835329996792
```





In [ ]:

In [ ]: