04_Exercise1_K-means

May 20, 2018

1 Team Members

1.1 Swaroop Bhandary K

1.2 Supriya Vadiraj

1.3 Vajra ganeshkumar

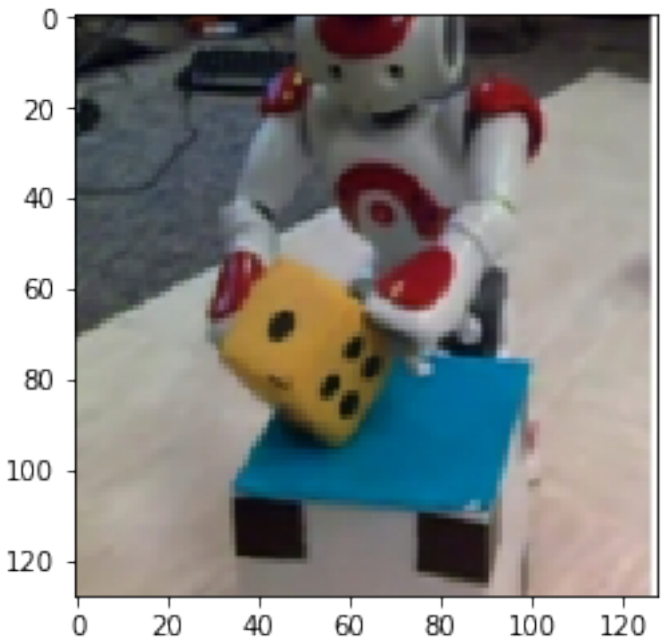

```
In [ ]: # Task: Image compression with K-means
```

Implement K-Means algorithm and apply it to compress an image "NAORelease.jpg" for various K (see slides for details). As a feature vector use RGB-representation of each pixel from the image. Analyse running time, what could you suggest to improve it? Compare your implementation with the existing k-mean algorithm given in python.

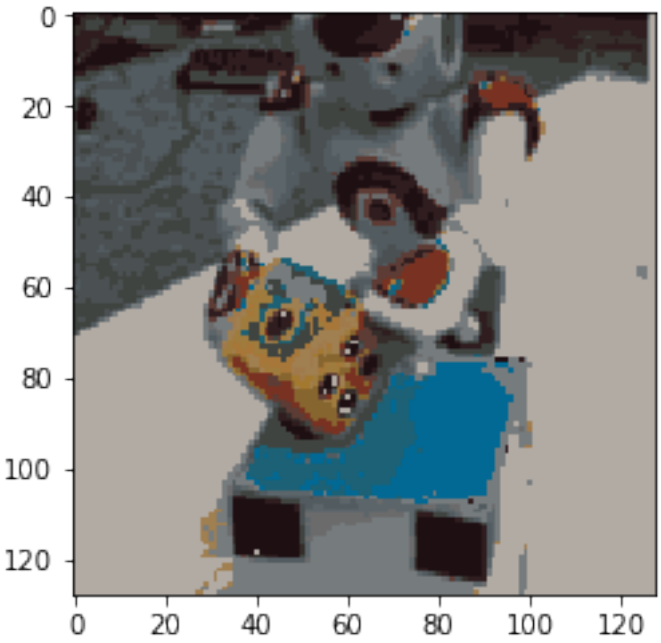
```
In [5]: from IPython.display import Image
        Image(filename='NAORelease.jpg')
        import numpy as np
        from PIL import Image
        import math
        from matplotlib import pyplot as plt
        import time
```

```
In [10]: def euclidean_distance(pixel_1, pixel_2):
             return math.sqrt(np.sum([(value[0]-value[1])**2 for value in zip(pixel_1, pixel_2)]

         image = np.array(Image.open('NAORelease.jpg'))
         number_of_cluster = 20

         start = [[np.random.randint(np.shape(image)[0]), np.random.randint(np.shape(image)[1])]
                  for i in range(number_of_cluster)]

         distance = np.zeros(number_of_cluster)
         points_cluster = {}

         plt.imshow(image, interpolation='nearest')
         plt.show()

         start_time = time.time()
```

```
for z in range(10):
    image = np.array(Image.open('NAORelease.jpg'))
    for i in range(number_of_cluster):
        points_cluster[i] = np.empty([0,5])

    for i in range(np.shape(image)[0]):
        for j in range(np.shape(image)[1]):
            distance = np.zeros(number_of_cluster)
            for k in range(number_of_cluster):
                distance[k] = euclidean_distance(image[start[k][0], start[k][1]], image
            cluster_no = np.argmin(distance)
            points_cluster[cluster_no] = np.vstack((points_cluster[cluster_no], np.hsta

    for i in range(number_of_cluster):
        single_cluster = points_cluster[i]
        if len(single_cluster) != 0:
            mean_value = np.mean(single_cluster, axis=0)
            start[i] = [int(mean_value[0]), int(mean_value[1])]
            for value in single_cluster:
                image[int(value[0]), int(value[1])] = mean_value[2:5]
        else:
            start[i] = [np.random.randint(np.shape(image)[0]), np.random.randint(np.sha

plt.imshow(image, interpolation='nearest')
plt.show()

print time.time()-start_time
```



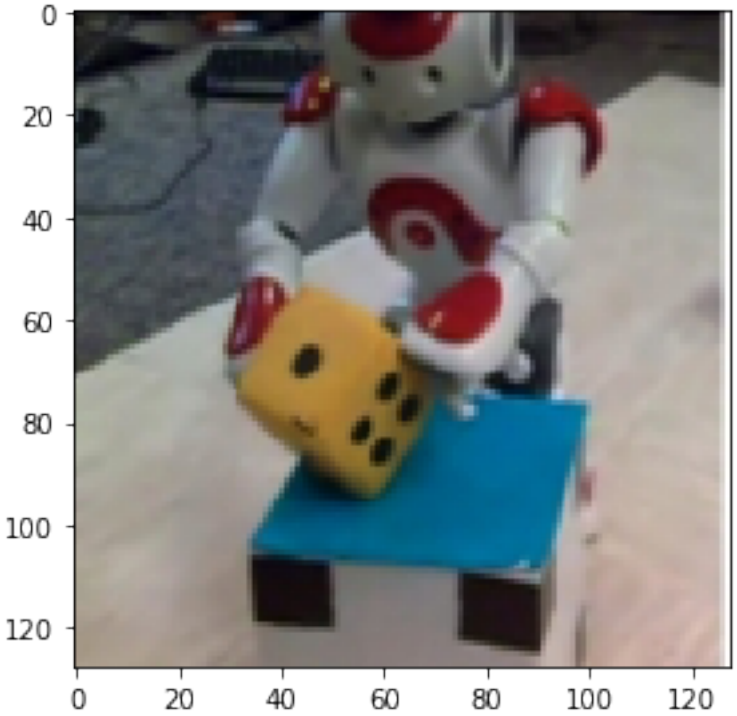
```
/home/swaroop/anaconda2/envs/tf-env/lib/python2.7/site-packages/ipykernel_launcher.py:2: Runtime
```




```
79.3348779678
```

```
In [15]: from skimage import io
         from sklearn.cluster import KMeans
         from sklearn.metrics import pairwise_distances_argmin
         from sklearn.utils import shuffle
         import numpy as np
         import matplotlib.pyplot as plt

         image = io.imread('NAORelease.jpg')
         n_colors = 18
         io.imshow(image)
         io.show()

         image = np.array(image, dtype=np.float64)/255

         rows = image.shape[0]
         columns = image.shape[1]
         image = image.reshape(image.shape[0]*image.shape[1],3)
```

```
image_array = shuffle(image, random_state=0)[:1000]
kmeans = KMeans(n_clusters=n_colors, random_state=0).fit(image_array)

labels = kmeans.predict(image)

def recreate_image(codebook, labels, rows, columns):
    """Recreate the (compressed) image from the code book & labels"""
    d = codebook.shape[1]
    img = np.zeros((rows, columns, d))
    label_idx = 0
    for i in range(rows):
        for j in range(columns):
            img[i][j] = codebook[labels[label_idx]]
            label_idx += 1
    return img

# Display all results, alongside original image

plt.figure(2)
plt.clf()
ax = plt.axes([0, 0, 1, 1])
plt.axis('off')
plt.title('Quantized image (64 colors, K-Means)')
plt.imshow(recreate_image(kmeans.cluster_centers_, labels, rows, columns))
```

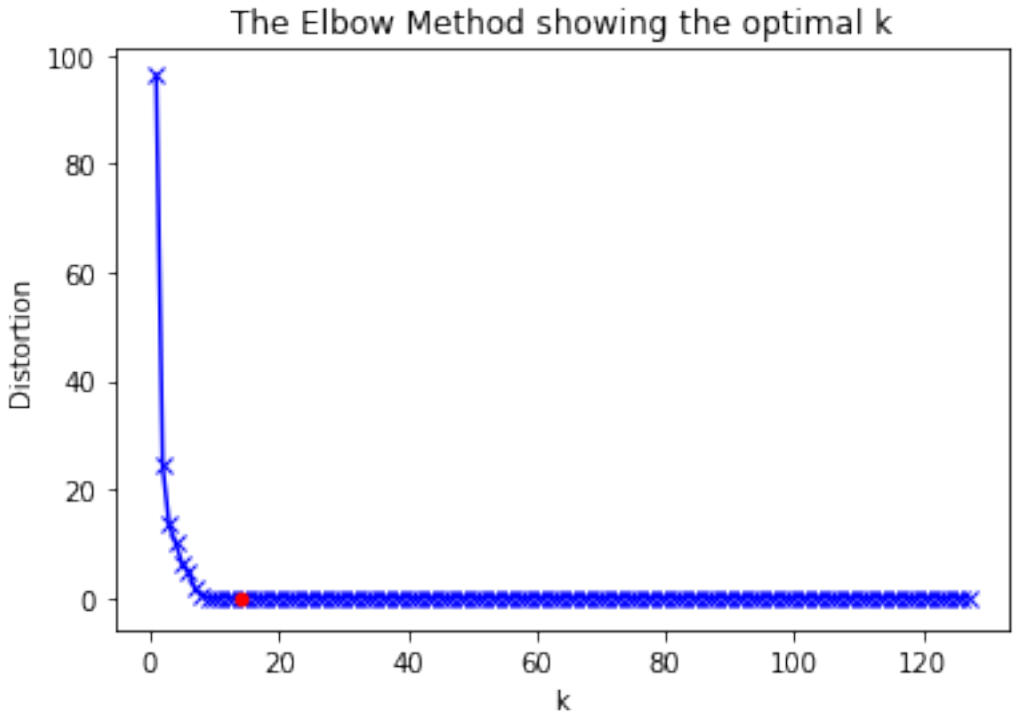


```
Fitting model on a small sub-sample of the data
Predicting color indices on the full image (k-means)
```

Out[15]: <matplotlib.image.AxesImage at 0x7fdb3eec3150>

In [20]:
```
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt

X = image[0]
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# k means determine k
distortions = []
```

```
K = range(1,128)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(X)
    kmeanModel.fit(X)
    distortions.append(sum(np.min(cdist(X, kmeanModel.cluster_centers_, 'euclidean'), a

# Plot the elbow
line2d = plt.plot(K, distortions, 'bx-', zorder = '1')
xvalues = line2d[0].get_xdata()
yvalues = line2d[0].get_ydata()
plt.scatter(xvalues[13],yvalues[13], c='r', s = 20, zorder = '2')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()

print yvalues.round(1)
print xvalues.round(1)
```




```
[96.3 24.3 13.4 10.   6.4  4.7  1.6  0.5  0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
```

```
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0. ]
[  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
 127]
```