

K-Means-based Image Segmentation

A Report on the performance of k-means-based image sgementation on the berkely segmentation benchmark dataset

Amr Mustafa – Mostafa Mahmoud | Pattern Recognition | Fall 2019

# The Dataset

The Berkeley segmentation benchmark dataset (BSR) contains a total of 500 images that are split into:

* A training/validation set of 300 images.
* A test set of 200 images.

All of our results are delivered on the test set.

Each image of the dataset was segmented by five different subjects on average to generate the ground truth data. A single example is shown below.



We can use the ground truth data to evaluate our segmentation.

Since the dataset comprises of colored images, we can view each pixel as a vector in the three-dimensional RGB color space. Using this representation, we can perform K-Means on the image easily as a collection of vectors.

# K-Means Algorithm

K-Means is a greedy iterative approach for clustering. It mainly consists of two steps:

1. Cluster assignment.
2. Centroid update.

The algorithm repeatedly performs the above steps until either it converges to a local minimum or it exceeds some specified maximum number of iterations. To make our program run faster, we decided to stick with specifying a number of iterations instead of waiting for convergence.

At the beginning, k random centroids are generated as follows:

u = np.random.randint(0, 256, (1, k, 3))

Note that we are generating three-dimensional centroids; one dimension for each feature of the pixel, i.e. each color axis.

In the cluster assignment step, we calculate the distance (L2 norm) between the current pixel vector and all the k centroids as follows:

d = [np.linalg.norm(m[j, :] - u[0][i])\*\*2

for j in range(m.shape[0]) for i in range(k)]

Once we have the norms, we can assign each pixel to the closest centroid as follows:

indicator = [np.argmin(d[i:i+k]) for i in range(0, len(d), k)]

Note that we do the assignment in a single step to generate an indicator vector for all the pixels of the image at once. Then we organize the clusters into a dictionary for convenience along with some information regarding the position of the pixel on the image.

for i in range(k):

for j in range(m.shape[0]):

if indicator[j] == i:

c[i].append(m[j, :])

p[j] = i

The last step in the K-Means algorithm is the centroid update step, in which we simply calculate the mean of each cluster and update the centroids with them.

for i in range(k):

if len(c[i]):

u[0][i] = np.sum(c[i], axis=0) / len(c[i])

# Results

We present the results of performing k-means image segmentation on some of the image in the test set. The number of clusters, k, is varied from 3 to 7 in the following example.

# Spectral Clustering vs. K-Means Clustering

K = 3

K = 5

K = 7

We used spectral clustering with a 5-NN affinity graph to cluster the same images we used k-means with. The results are given below.

The results on the left side of the page are from the spectral clustering algorithm, whereas the results of the right side are the results from our k-means algorithm on the same set of images in order to compare and contrast the performance of the two algorithms qualitatively. All results are generated at k = 5.







