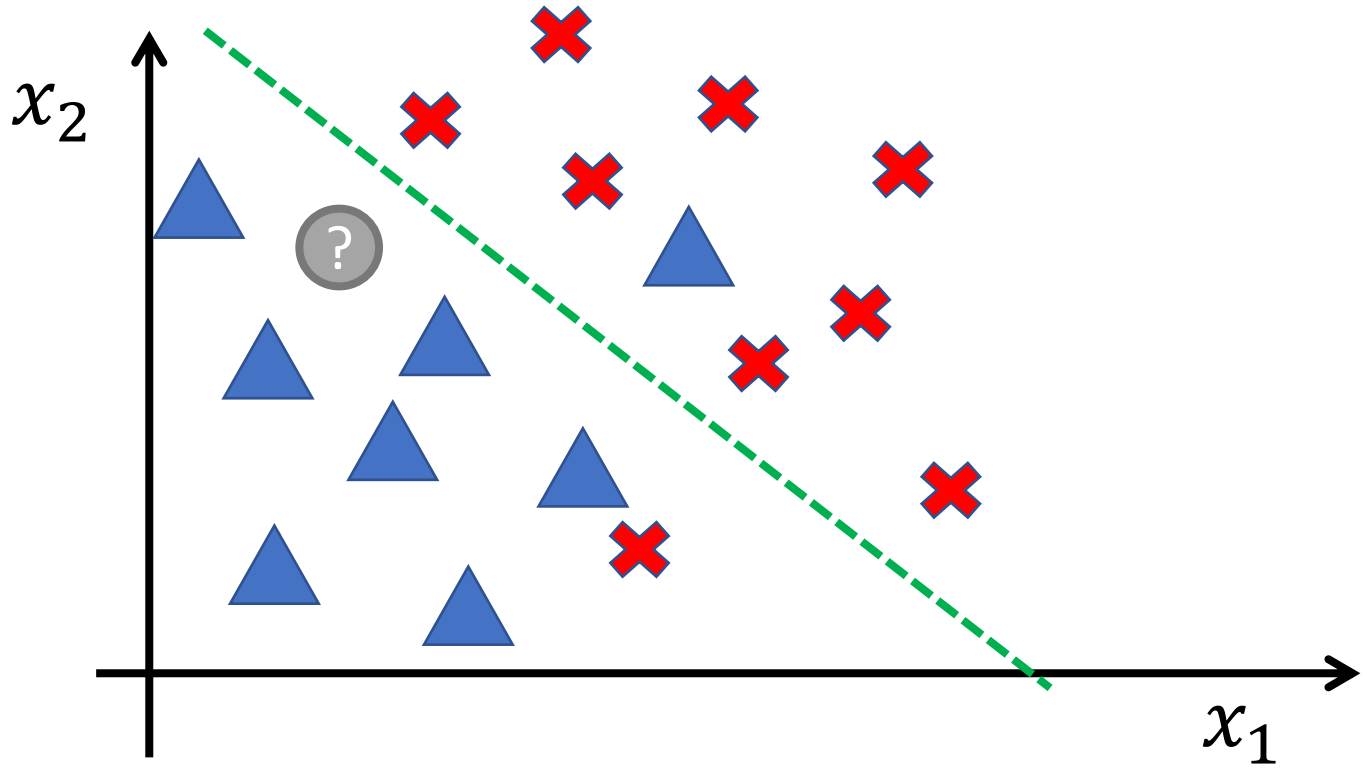
Clustering

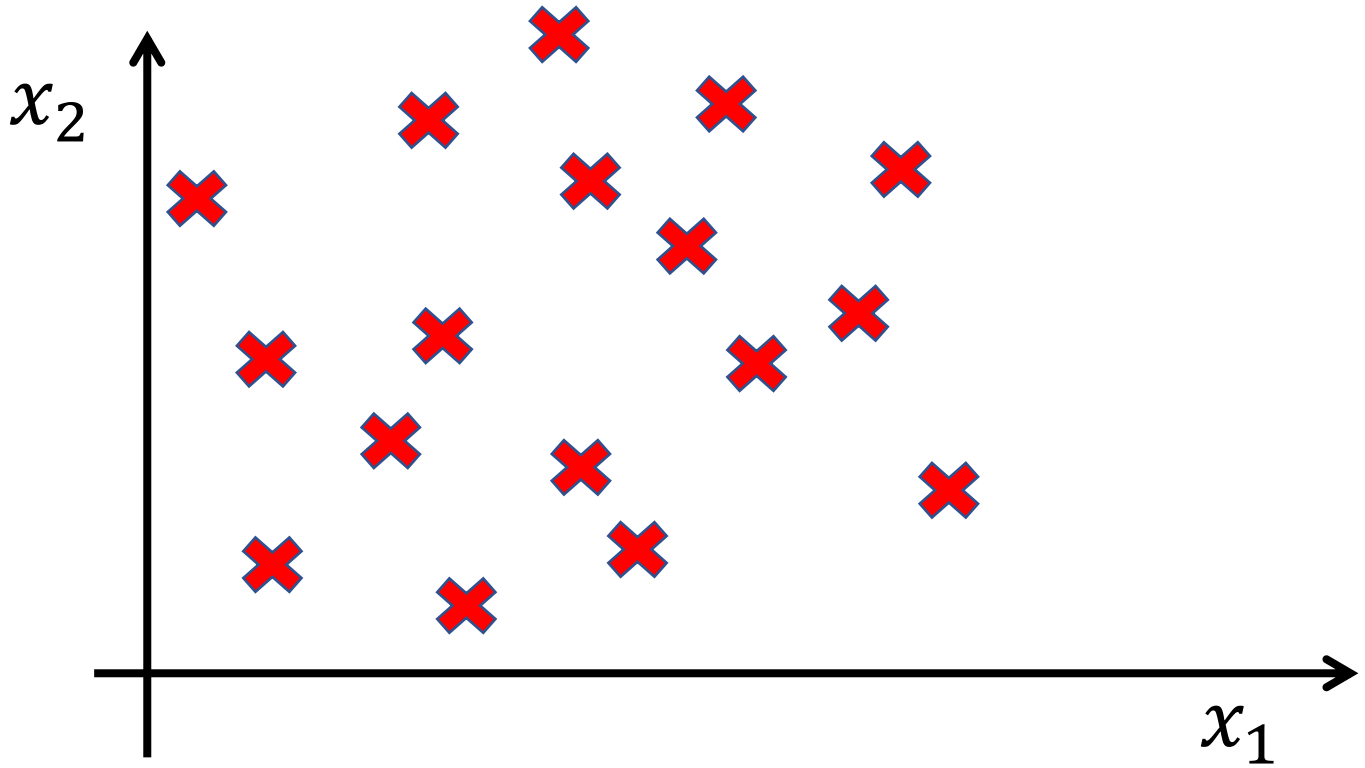
Supervised learning



- Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

Slide credit: Andrew Ng

Unsupervised learning



- Training set: $\{x^{(1)}, x^{(2)}, x^{(3)}, \cdots, x^{(m)}\}$

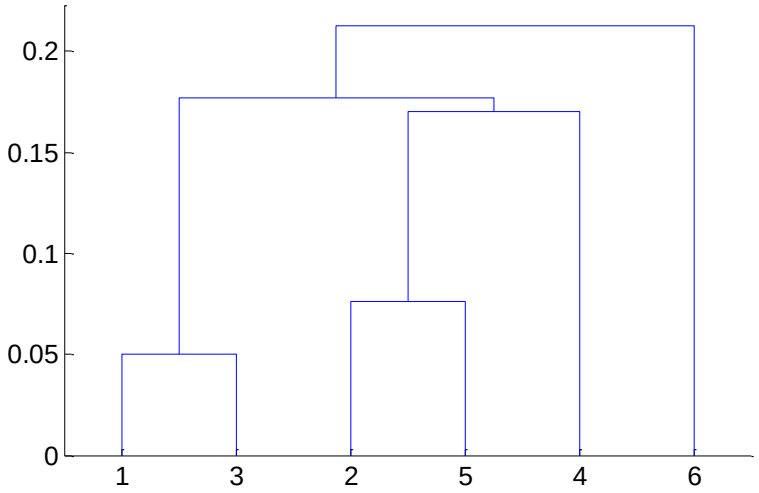
Slide credit: Andrew Ng

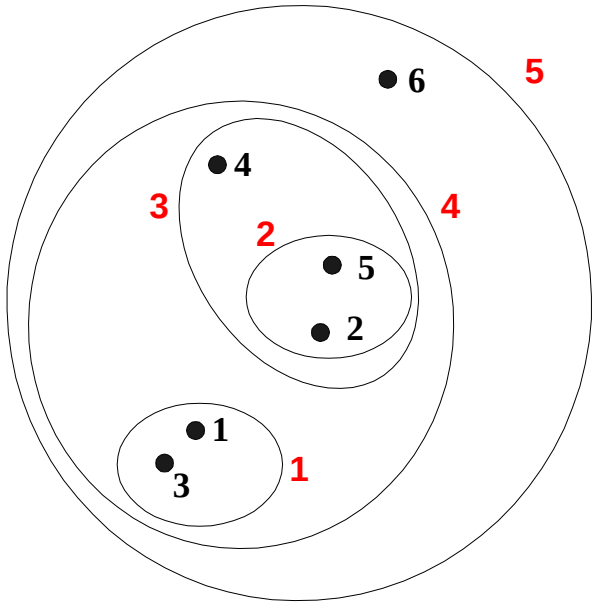
Clustering Applications

- **Customer Segmentation**: Businesses use clustering to group customers with similar purchasing behavior, demographics, or preferences. This helps in targeted marketing and personalized services.
- **Image Segmentation:** Clustering is used to segment images into meaningful regions based on color, texture, or other visual features.
- **Anomaly Detection:** Clustering helps identify unusual patterns or outliers in datasets. Any data points that do not fit well into any cluster may be considered anomalies, helping in detecting fraud, errors, or unusual behavior.
- And More….

Hierarchical Clustering

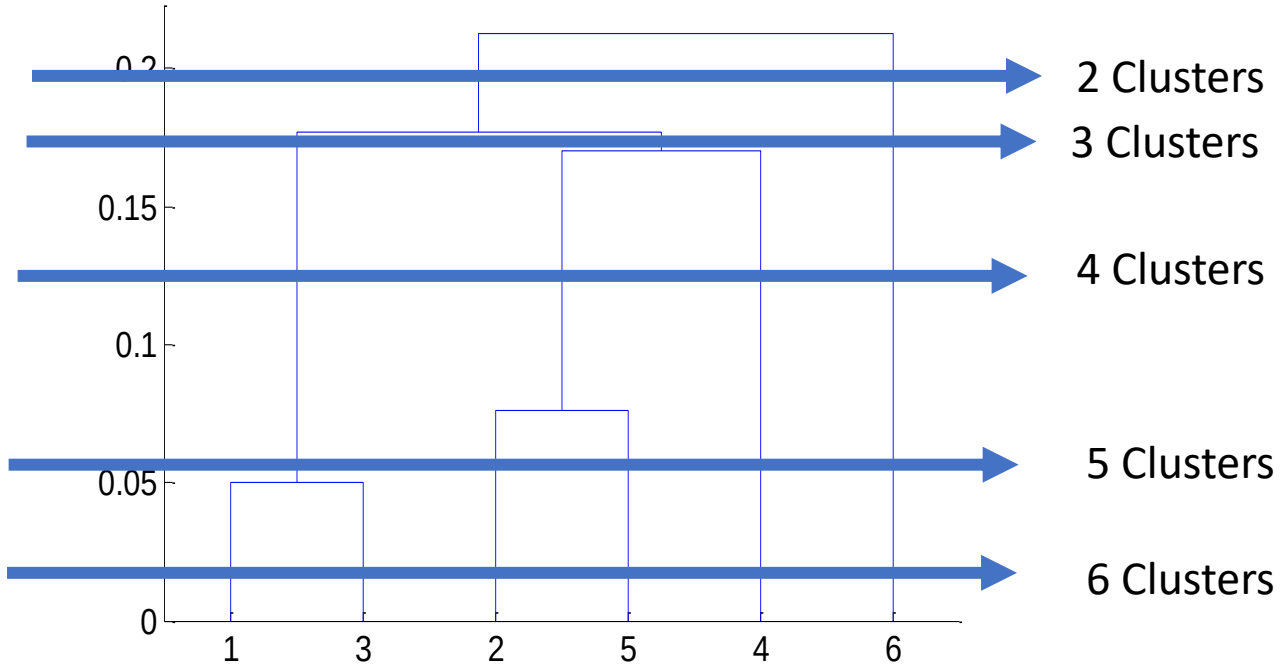
- Produces a set of ***nested clusters*** organized as a hierarchical tree
- Can be visualized as a **dendrogram**
  - A tree-like diagram that records the sequences of merges or splits





Strengths of Hierarchical Clustering

- No assumptions on the number of clusters
  - Any desired number of clusters can be obtained by ‘cutting’ the dendrogram at the proper level



Hierarchical Clustering

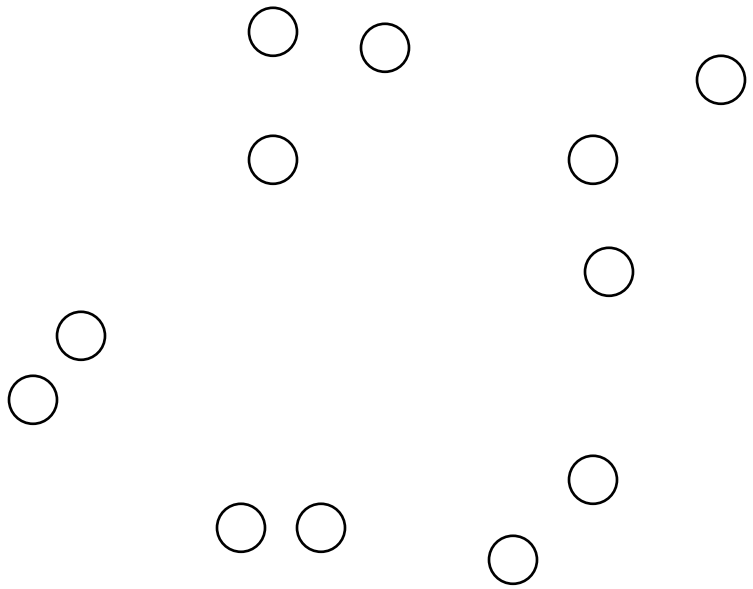
- Two main types of hierarchical clustering
  - **Agglomerative:**
    - Start with the points as individual clusters
    - At each step, merge the closest pair of clusters until only one cluster (or **k** clusters) left
  - **Divisive:**
    - Start with one, all-inclusive cluster
    - At each step, split a cluster until each cluster contains a point (or there are **k** clusters)
- Traditional hierarchical algorithms use a similarity or distance matrix
  - Merge or split one cluster at a time

Agglomerative clustering algorithm

- Most popular hierarchical clustering technique
- **Basic algorithm**
  1. Compute the distance matrix between the input data points
  2. Let each data point be a cluster
  3. **Repeat**
  4. Merge the two closest clusters
  5. Update the distance matrix
  6. **Until** only a single cluster remains
- Key operation is the computation of the distance between two clusters
  - Different definitions of the distance between clusters lead to different algorithms

Input/ Initial setting

- Start with clusters of individual points and a distance/proximity matrix



	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						

Distance/Proximity Matrix

Distance between two clusters

- Each cluster is a set of points
- How do we define distance between two sets of points?
  - Lots of alternatives

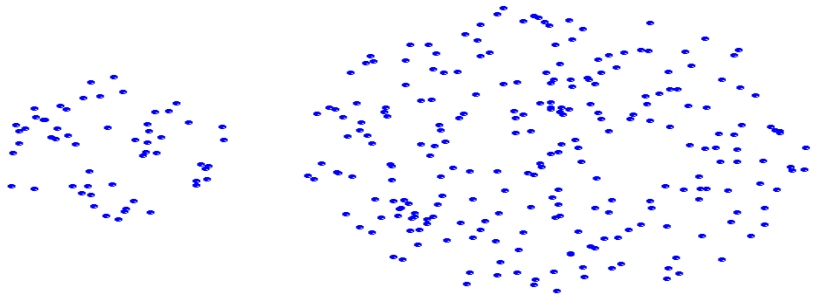
Distance between two clusters

- **Single-link distance** between clusters C_i and C_j is the ***minimum distance*** between any object in C_i and any object in C_j

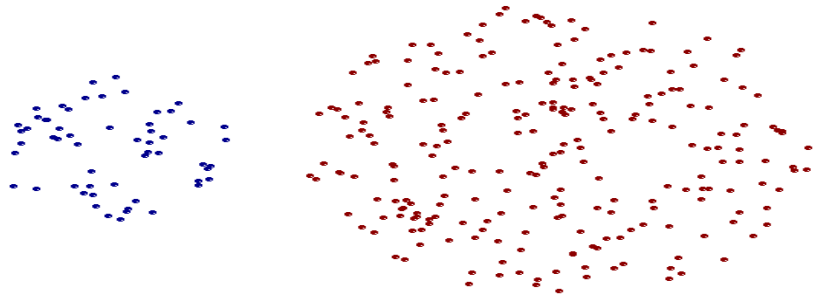
- The distance is **defined by the two most similar objects**

$$D_{sl}(C_i, C_j) = \min_{x,y} \left\{ d(x, y) \middle| x \in C_i, y \in C_j \right\}$$

Strengths of single-link clustering

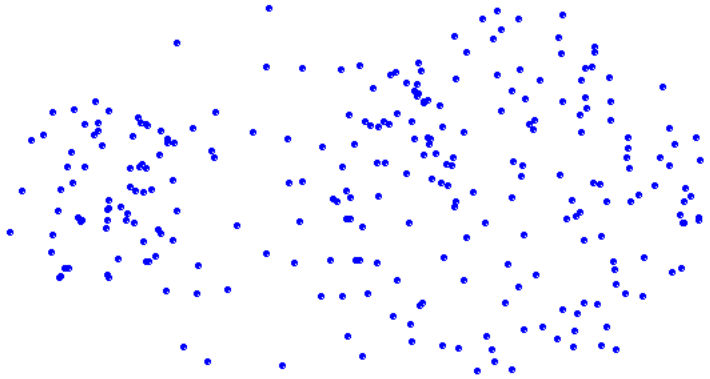


Original Points

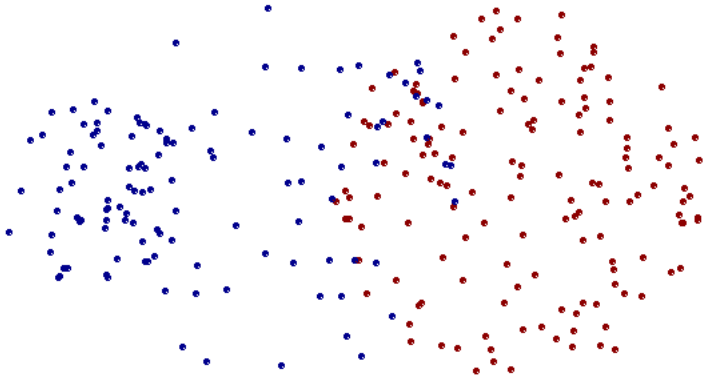


Two Clusters

Limitations of single-link clustering



Original Points



Two Clusters

- **Sensitive to noise and outliers**
- **It produces long, elongated clusters**

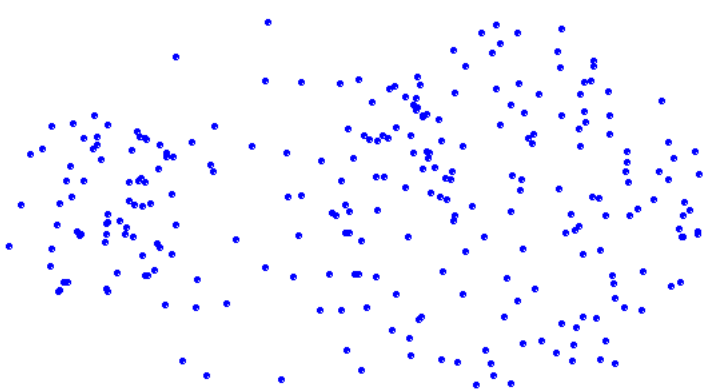
Distance between two clusters

- **Complete-link distance** between clusters C_i and C_j is the ***maximum distance*** between any object in C_i and any object in C_j

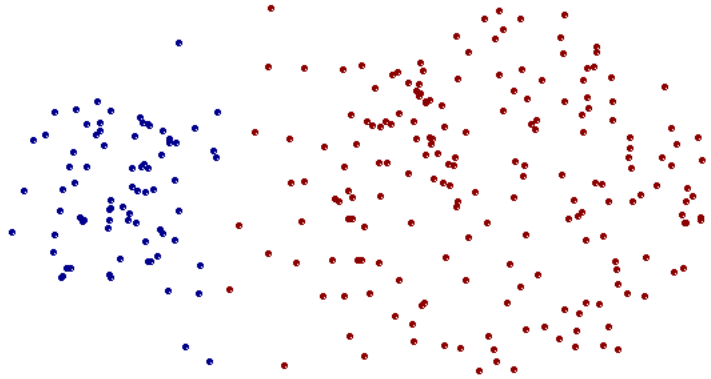
- The distance is **defined by the two most dissimilar objects**

$$D_{cl}(C_i, C_j) = \max_{x,y} \{ d(x, y) \mid x \in C_i, y \in C_j \}$$

Strengths of complete-link clustering



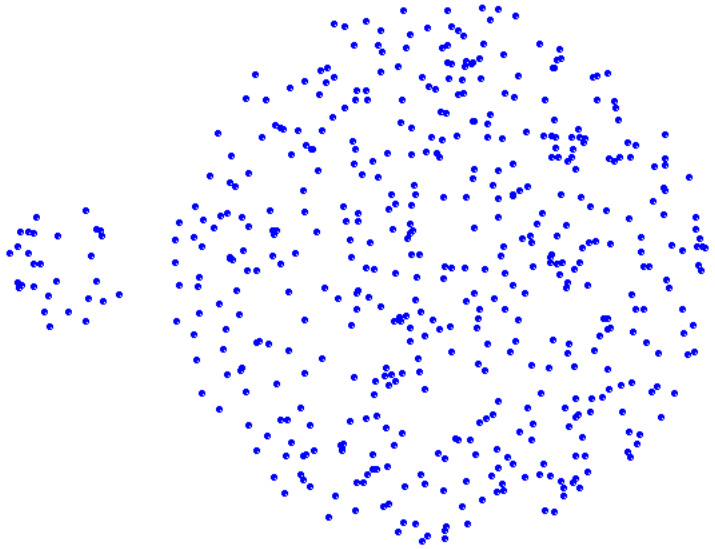
Original Points



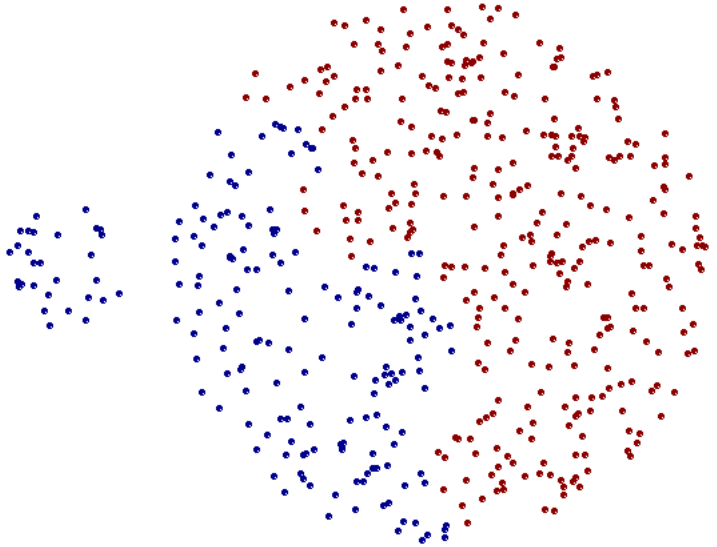
Two Clusters

- **Less susceptible to noise**

Limitations of complete-link clustering



Original Points



Two Clusters

- **Tends to break large clusters**
- **All clusters tend to have the same diameter – small clusters are merged with larger ones**

Distance between two clusters

- **Group average distance** between clusters C_i and C_j is the ***average distance*** between any object in C_i and any object in C_j

$$D_{avg}(C_i, C_j) = \frac{1}{|C_i| \times |C_j|} \sum_{x \in C_i, y \in C_j} d(x, y)$$

Average-link clustering: discussion

- Compromise between Single and Complete Link
- Strengths
  - Less susceptible to noise and outliers
- Limitations
  - Biased towards globular clusters

Distance between two clusters

- **Centroid distance** between clusters C_i and C_j is the distance between the centroid r_i of C_i and the centroid r_j of C_j

$$D_{centroids}(C_i, C_j) = d(r_i, r_j)$$

Distance between two clusters

- **Ward's distance** between clusters C_i and C_j is the ***difference*** between the ***total within cluster sum of squares for the two clusters separately***, and the ***within cluster sum of squares resulting from merging the two clusters*** in cluster C_{ij}

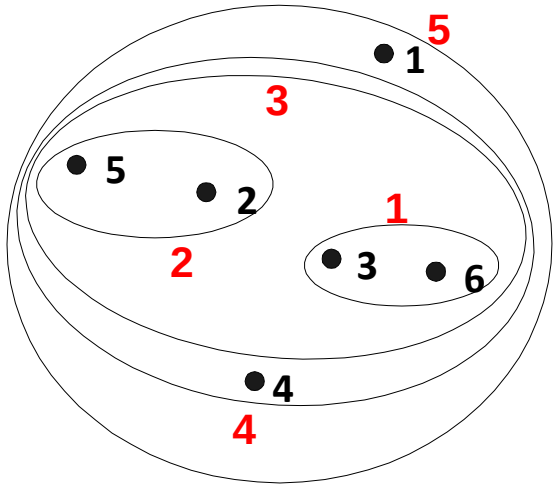
$$D_w(C_i, C_j) = \sum_{x \in C_i} (x - r_i)^2 + \sum_{x \in C_j} (x - r_j)^2 - \sum_{x \in C_{ij}} (x - r_{ij})^2$$

- r_i: centroid of C_i
- r_j: centroid of C_j
- r_{ij}: centroid of C_{ij}

Ward’s distance for clusters

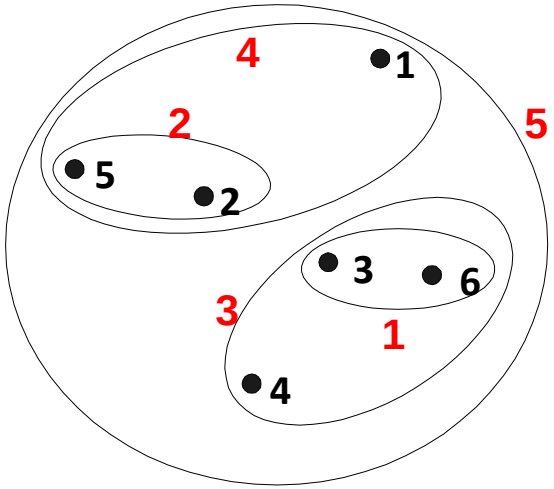
- Similar to group average and centroid distance
- Less susceptible to noise and outliers
- Biased towards globular clusters

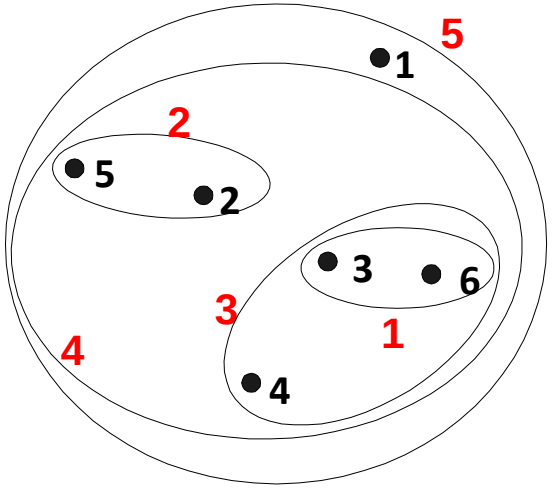
Hierarchical Clustering: Comparison



MIN

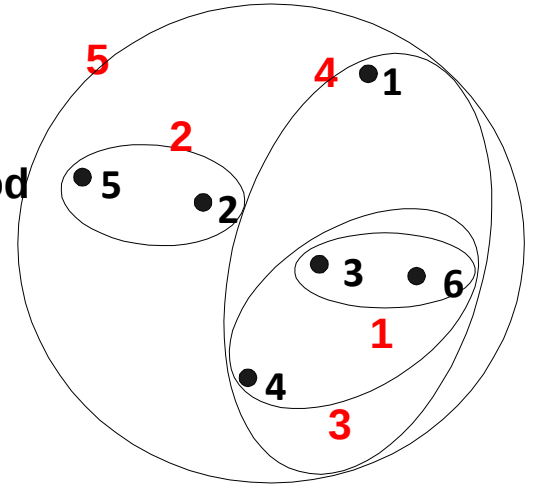
MAX





Group Average

Ward's Method



K-means Clustering

K-means Clustering

- K-means algorithm is used to cluster data that share similar features into several classes.
- Each class is identified by its class center (centroid). The goal is to find these sets of class centers that minimize a certain error criterion.
- For color images, the cluster center is a 3D color vector for each class k of the assumed number of classes K. For example, $C_k=\{C_{kr}, C_{kg}, C_{kb}\}$, where $k = 1, \ldots, K$ and C_{kr}, C_{kg}, and C_{kb} are the RGB components of the mean color of pixels that belong to cluster k.

Algorithm: K-means Clustering

- The following are the steps to implement the k-means algorithm for color images:

  I. Initialization: determine the number of clusters K, then initialize the cluster centers C_k. The initialization can simply be random values.

  II. Assignment: for each input pixel p(x,y) = {p_r, p_g, p_b}, find the winner cluster k, whose cluster center C_k has the smallest distance to the input pixel. This distance can be calculated using the Euclidean distance as:

$$E_k = \sqrt{(p_r - C_{kr})^2 + (p_g - C_{kg})^2 + (p_b - C_{kb})^2}$$

Add pixel p to the set S_k of the winner class k. Repeat for each pixel of the image.

Algorithm: K-means Clustering

III. Update: for each cluster find the number of pixels, N_k, assigned to set S_k. Find the new cluster centers, C_k^{new} for each cluster as:

$$C_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N_k} p_k, where \;\; pk \in Sk$$

iv. Repeat step II and III. Stop when C_k didn't change or update significantly between iterations.

Code

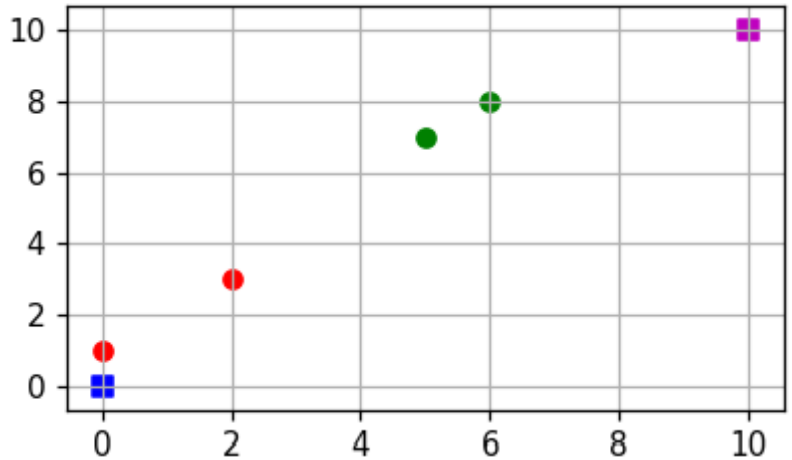
In [1]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import cv2
import sys
```

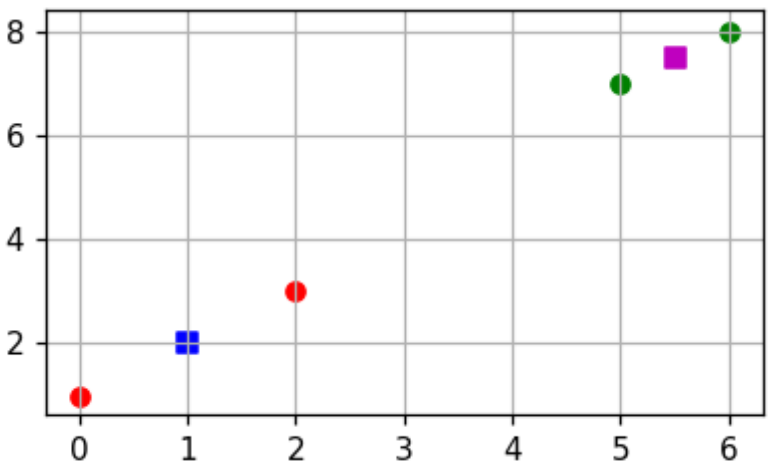
In [2]:
```
def kmeans(in_data, centers, num_clusters, num_iter):
    s = np.shape(in_data)
    label = np.zeros(s[0])
    for t in range(1, num_iter + 1): #iterations
        print('Iteration #: ', t)
        dist = np.zeros([s[0], num_clusters]) # distance measure for each cluster
        mn = sys.maxsize # minimum distance
        for i in range(0, s[0]): # for each data point
            for k in range(0, num_clusters): # for each cluster
                # find the distance between data point and each cluster center
                dist[i, k] = np.mean(np.abs(in_data[i,:] - centers[k,:]))
            mn = np.min(dist[i,:]) # find the minimum distance
            # find which cluster has this minimum distance
            cluster = list(filter(lambda x: dist[i, x] == mn, range(num_clusters)))
            label[i] = cluster[0] # label this data point with this cluster
        # update cluster centers by averaging data points that assigned to the same cluster
        for k in range(0, num_clusters):
            L = list(filter(lambda x: label[x] == k, range(s[0]))) # indices of data points that share the same cluster
            if (L): # should not be empty
                centers[k, :] = np.mean(in_data[L,:], axis = 0) # average data
            #centers[k, :] = np.mean(in_data[label == k,:], axis = 0) # average data
    return np.int32(label), centers
```

Example_1

```
in_data = np.array([[0,1],[2,3],[5,7],[6,8]],dtype = np.float)
print("Input Data:\n", in_data)
num_iter = 2
num_clusters = 2
centers = np.array([[0,0],[10,10]],dtype = np.float)
init_centers = np.copy(centers)
print('\nInitial Centers\n', init_centers)
label, centers = kmeans(in_data, centers, num_clusters, num_iter)
print('\nlabels:\n', label,'\n','\nCluster Centers\n', centers)
```

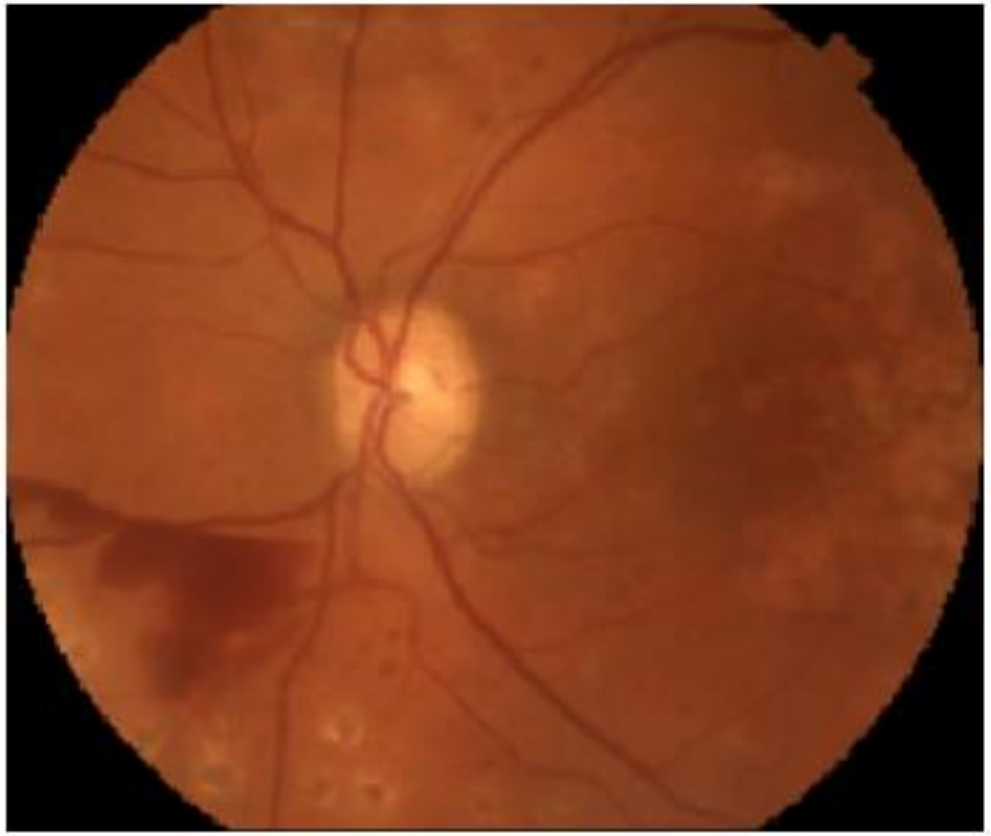
```
Input Data:
 [[0. 1.]
 [2. 3.]
 [5. 7.]
 [6. 8.]]

Initial Centers
 [[ 0.  0.]
 [10. 10.]]


 labels:
  [0 0 1 1]

 Cluster Centers
  [[1.  2. ]
  [5.5 7.5]]
```





Example_2

Use K-means to cluster the below fundus image to four classes.

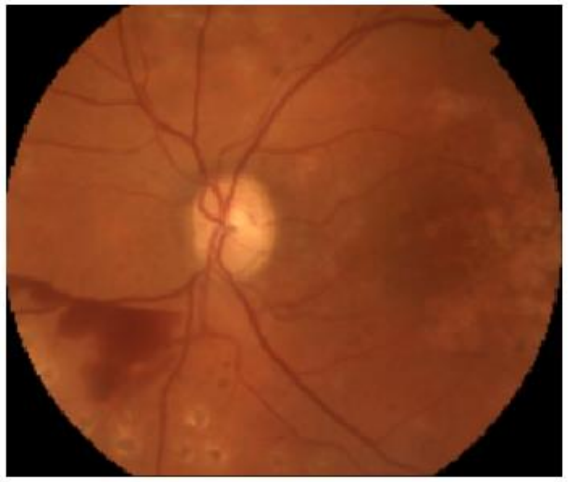


Results

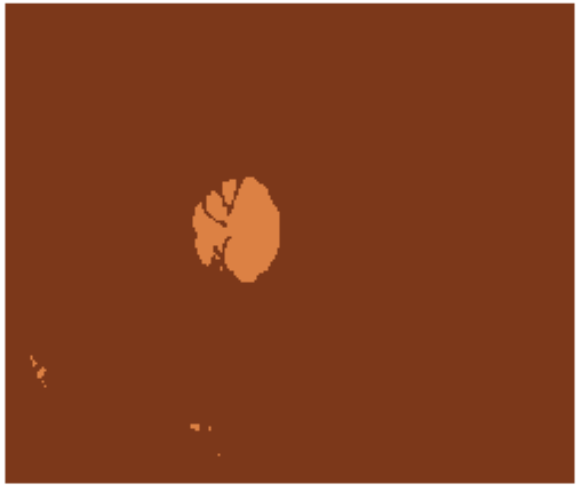
```
img = cv2.imread('fundus_1.png')
in_data = np.float32(img.reshape((-1,3)))
num_iter = 10
num_clusters = 4
np.random.seed(0)
centers = np.random.uniform(0,255,(num_clusters,3))
print(centers)
label, class_centers = kmeans(in_data, centers, num_clusters, num_iter)
out = np.uint8(class_centers[label])
out2 = out.reshape(np.shape(img))
```

Initial BGR cluster centers

```
[[139.9474435  182.37328842 153.7046609 ]
 [138.94521166 108.03197383 164.70299883]
 [111.58473887 227.4021152  245.73400393]
 [ 97.7775873  201.88988471 134.86820454]]
```

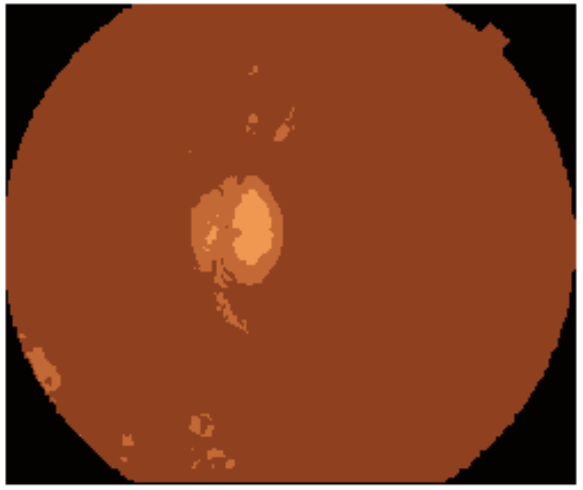


input



#iterations = 2

#iterations = 5

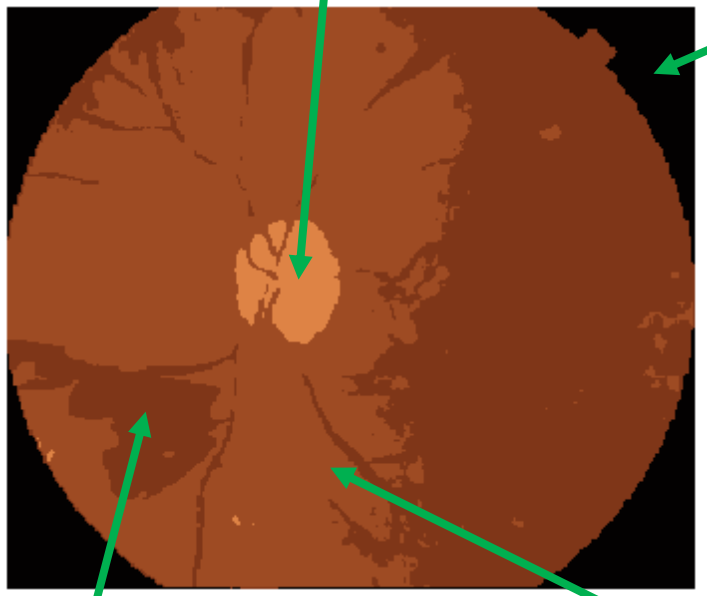


#iterations = 10

Results

```
plt.figure(2)
plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(cv2.cvtColor(out2,cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

input



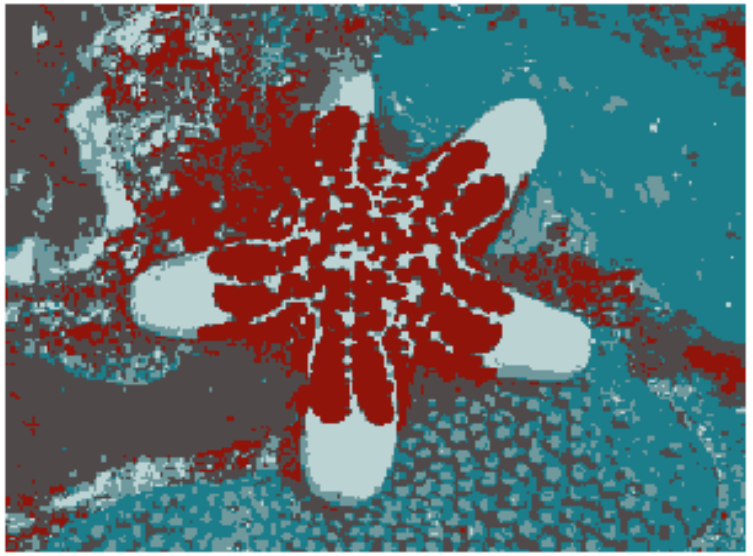
#iterations = 20

```
[[ 35  75 158]
 [  1   1   3]
 [ 24  54 127]
 [ 69 131 222]]
```

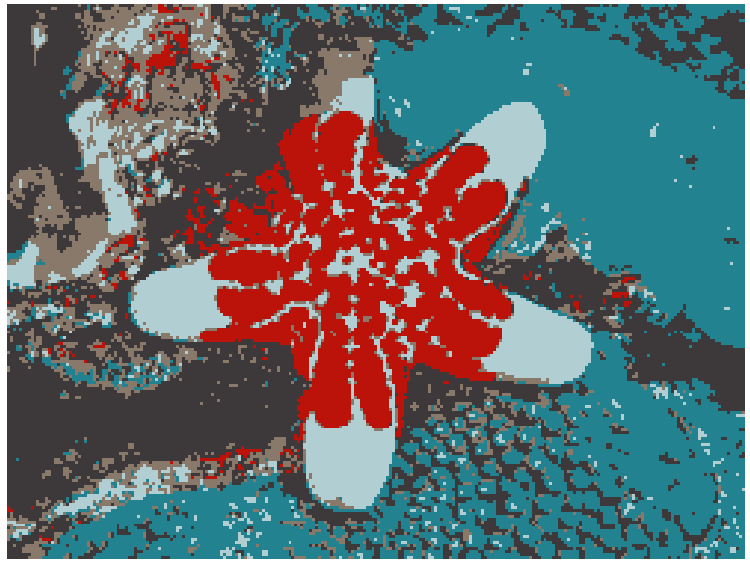
More Results: K-means Clustering

K-means results in RGB color space with different number of iterations

input



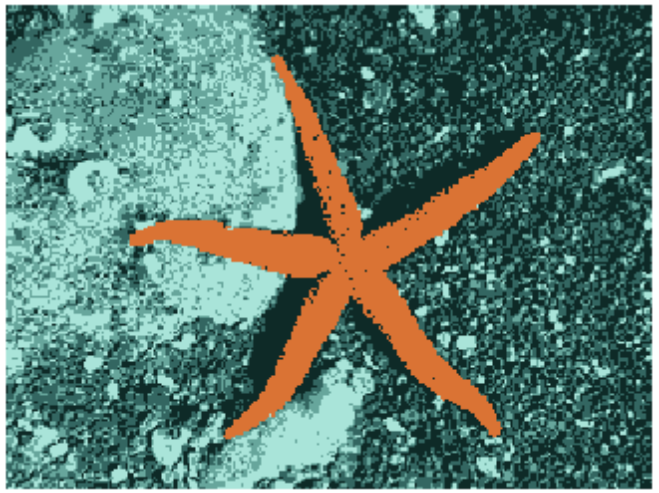
#iterations = 10



#iterations = 20

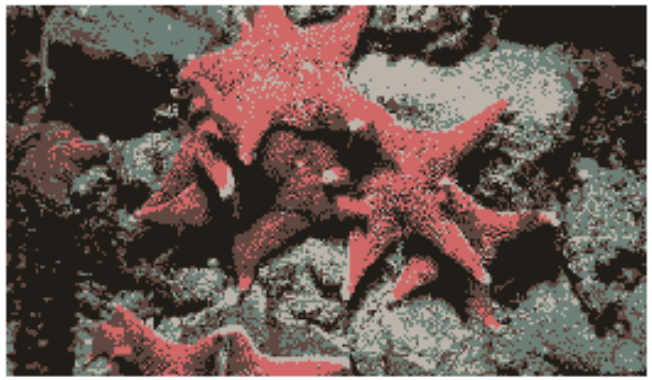
More Results: K-means Clustering

K-means results in RGB color space



BGR cluster centers

```
[[217 228 169]
 [ 97 102  51]
 [ 52 115 218]
 [156 166 103]
 [ 39  42  14]]
```



BGR cluster centers

```
[[170 180 188]
 [ 64  72  96]
 [104 104 209]
 [119 128 109]
 [ 23  28  32]]
```

input

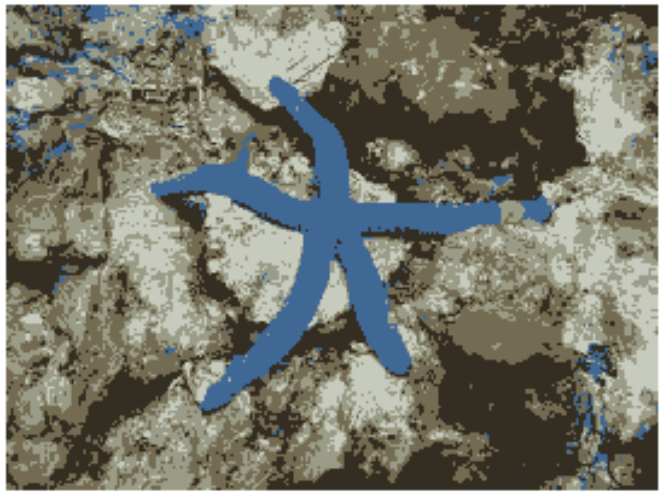
K-means clustering output

More Results: K-means Clustering

K-means results in RGB

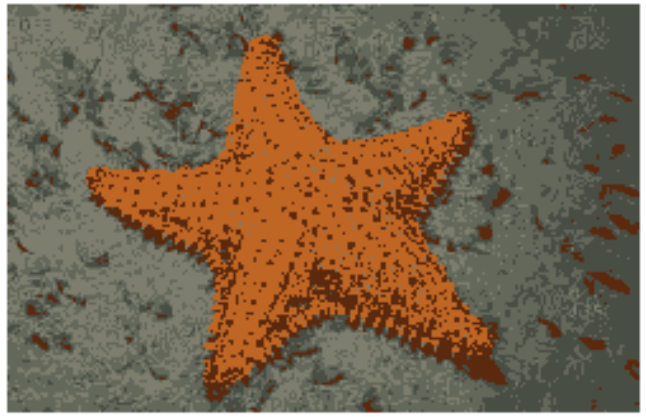
input

K-means clustering output



BGR cluster centers

```
[[135 156 155]
 [149 104  64]
 [189 203 198]
 [ 82 108 116]
 [ 33  45  52]]
```

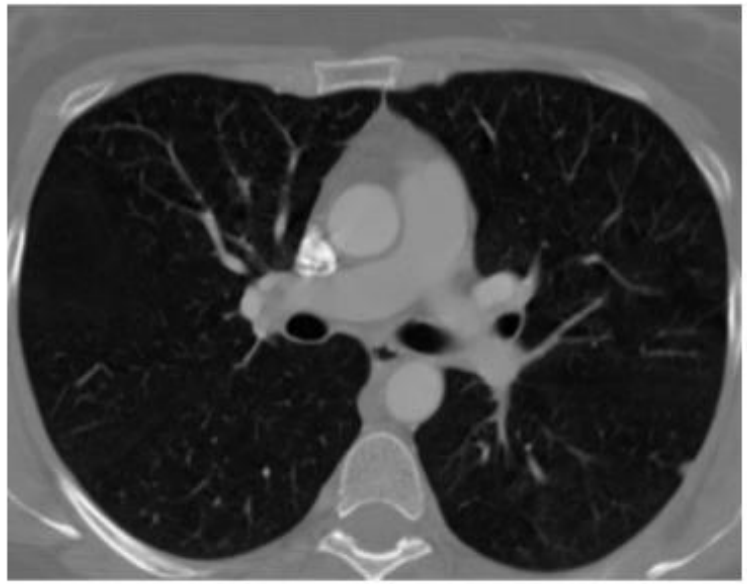


BGR cluster centers

```
[[110 126 125]
 [ 69  78  73]
 [ 37 102 191]
 [ 91 104 100]
 [ 14  41  91]]
```

Example_3

Use K-means to cluster the below Chest images to two classes.



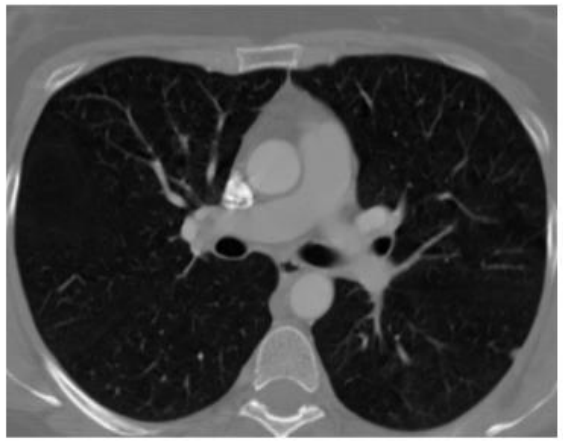
Code

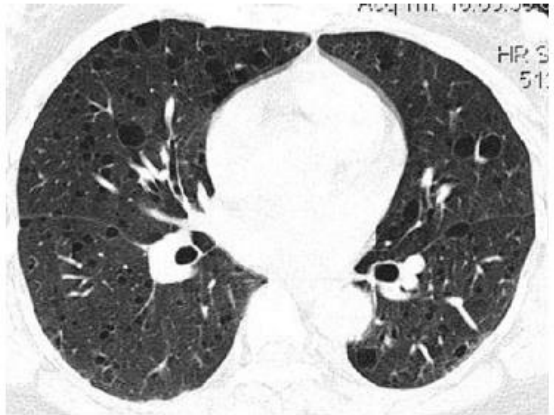
```
#img = cv2.imread('lung_crop.pgm',cv2.IMREAD_GRAYSCALE)
img = cv2.imread('Chest_CT_IM.png',cv2.IMREAD_GRAYSCALE)
print(np.ndim(img))
in_data = np.float32(img.reshape((-1,1)))
num_iter = 20
num_clusters = 2
np.random.seed(0)
centers = np.random.uniform(0,255,(num_clusters,1))
print(centers)
label, class_centers = kmeans(in_data, centers, num_clusters, num_iter)
out = np.uint8(class_centers[label])
out2 = out.reshape(np.shape(img))
```

```
plt.figure(5)
plt.subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(out2,cmap='gray')
plt.axis('off')
plt.show()
```

Results

Input





Output

Cluster Centers:
[[89]
[155]]

Cluster Centers:
[[74]
[243]]