Introduction

- K-means is unsupervised learning and it clusters the dataset such that each cluster has the same properties
- Based on the input and number of cluster, determine the centroid of each cluster and classify each data point
- Assume we have N data points in training set $X \in \mathbb{R}^{N\times d}$ (N data points and d feature) and $K < N$ is number of cluster. We need to find out centroid $\mathbf{m}_1, \mathbf{m}_2, \ldots \mathbf{m}_K \in \mathbb{R}^d$ and label of each point

Math aspects

- For each data point x_i, we need to find $y_i = k$ in which $k \in \{1, 2, \ldots, K\}$
- Instead of using category for label, we can use One-Hot Encode. E.g. if point has label [1, 0, 0, …, 0], so it is in label 1, if point has label [0, 1, 0, …, 0], so it is in label 2, so $y_{ij} \in \{0,1\},\ \forall i, j;\ \sum_{j=1}^{K} y_{ij} = 1$

Cost function and optimization

- Assume $\mathbf{m}_k \in \mathbb{R}^d$ is the centroid of the cluster k, one data point x_i is classified into cluster k
- What we want is to minimize this distance between x_i and m_k, which means $\min \|\mathbf{x}_i - \mathbf{m}_k\|_2^2$, because x_i is classified into cluster k, $y_{ik} = 1$, $y_{ij} = 0$, $\forall j \neq k$

$$\|\mathbf{x}_i - \mathbf{m}_k\|_2^2 = y_{ik} \|\mathbf{x}_i - \mathbf{m}_k\|_2^2 = \sum_{j=1}^{K} y_{ij} \|\mathbf{x}_i - \mathbf{m}_k\|_2^2$$

- Cost function:

$$L(Y, M) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \|\mathbf{x}_i - \mathbf{m}_k\|_2^2 \rightarrow Y, M = \arg\min_{Y,M} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

in which $M = \begin{bmatrix} \mathbf{m}_1^T & \mathbf{m}_2^T & \ldots & \mathbf{m}_K^T \end{bmatrix} \in \mathbb{R}^{K\times d}$

- We have 2 parameters to optimize, so one technique should be used: Assume Y is constant, optimize M and assume M is constant, optimize Z
  - Assume M is constant, optimize Z: Assume we already found the centroids, classify the data points such that cost function is min

$$y_i = \arg\min_{y_i} \frac{1}{N} \sum_{j=1}^{K} y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

  - Because $\sum_{j=1}^{K} y_{ij} = 1$ which means there is only one $y_{ij} = 1$, so we just find which centroid j x_i is nearest and put $y_{ij} = 1$, so

$$j = \arg\min_{j} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

  - Assume Y is constant, optimize M: Assume we already found the cluster of each point, find the new centroids for each cluster

$$m_j = \arg\min_{m_j} \frac{1}{N} \sum_{i=1}^{N} y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

  - Take derivative to find min (we can also take derivative for first assumption):

$$\nabla_{m_j} l(m_j) = \frac{2}{N}\sum_{i=1}^{N} y_{ij}\left(\mathbf{m}_j - \mathbf{x}_i\right) = 0 \rightarrow \mathbf{m}_j \sum_{i=1}^{N} y_{ij} = \sum_{i=1}^{N} y_{ij}\mathbf{x}_i \rightarrow \mathbf{m}_j = \frac{\sum_{i=1}^{N} y_{ij}\mathbf{x}_i}{\sum_{i=1}^{N} y_{ij}}$$

- So, m_j is the mean of all data points in cluster j

Hierarchical Clustering

There are 2 types:

- **Agglomerative**: first we choose K ≈ amount of data points, after first KMeans clustering, the close clusters will be merged into a cluster, so we will get smaller number of clusters than K clusters. Iterate this process until the desired number of cluster
- **Divisive**: first we consider dataset as one cluster, then separate the dataset by clustering algorithm like KMeans. In each cluster we got, we apply cluster algorithm to separate. Iterate this process until the desired number of cluster

Image Compression

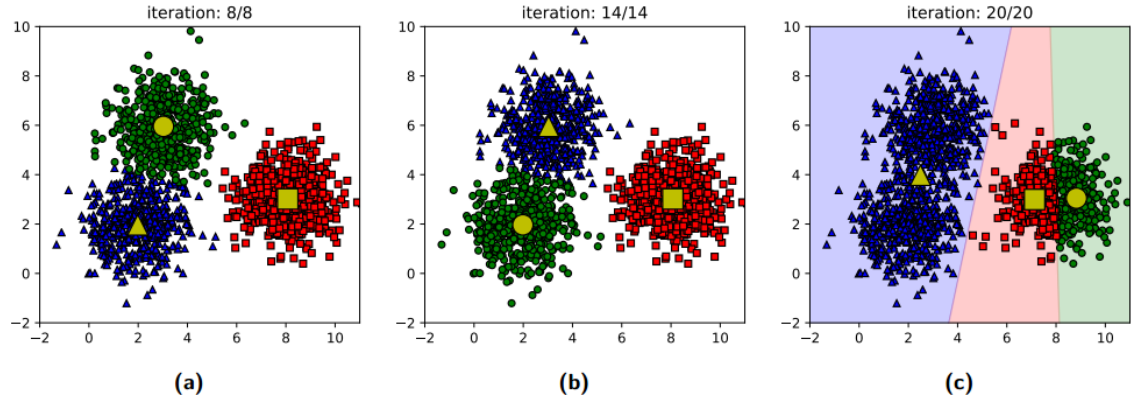
- Consider this code

```
for K in [2, 5, 10, 15, 20]:
    kmeans = KMeans(n_clusters=K).fit(X)
    label = kmeans.predict(X)

    processed_img = np.zeros_like(X)
    for i in range(K):
        processed_img[label == i] = kmeans.cluster_centers_[i]

    processed_img = processed_img.reshape(img.shape[0], img.shape[1], img.shape[2])
    plt.title('KMeans with ' + str(K) + ' clusters')
    plt.imshow(processed_img, interpolation='nearest')
    plt.axis('off')
```

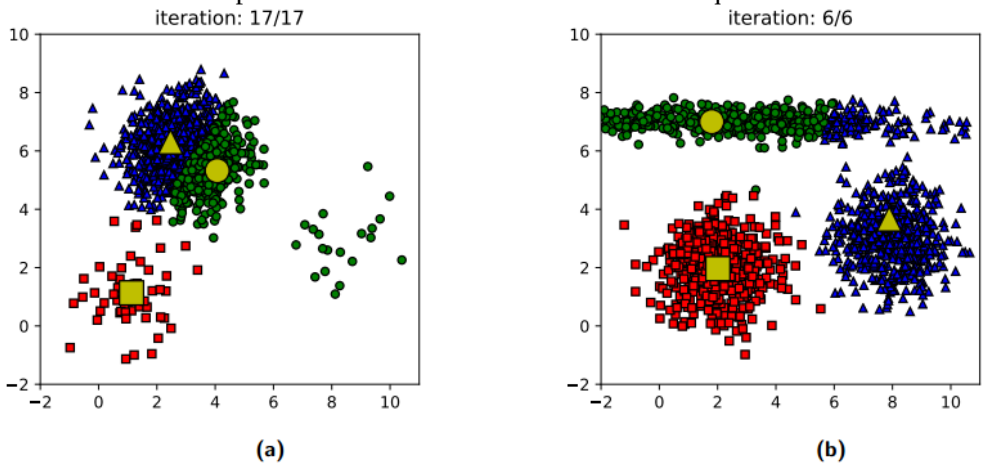
- After finding the centroids, the value of each pixel is assigned to its centroid, so the larger the K is, the better the quality of processed image is

Drawbacks

- Have to determine K clusters: to get the optimal K, we need to cross validate. **Elbow Method** can be applied here
- The final centroids depend on initial centroid which are randomly initiated: K-Means does not guarantee finding the global optimal centroid, so it depends on the initial centroid we choose. Some proof
  - Final centroids of image (a) and (b) is good, but image (c) is quite bad. Additionally, implementation time of image (c) is twice more than (a) and (b)
  - KMeans++ may be helpful to solve above problem



- Total number of data point in each cluster need to be almost equal: we should choose other initial centroids

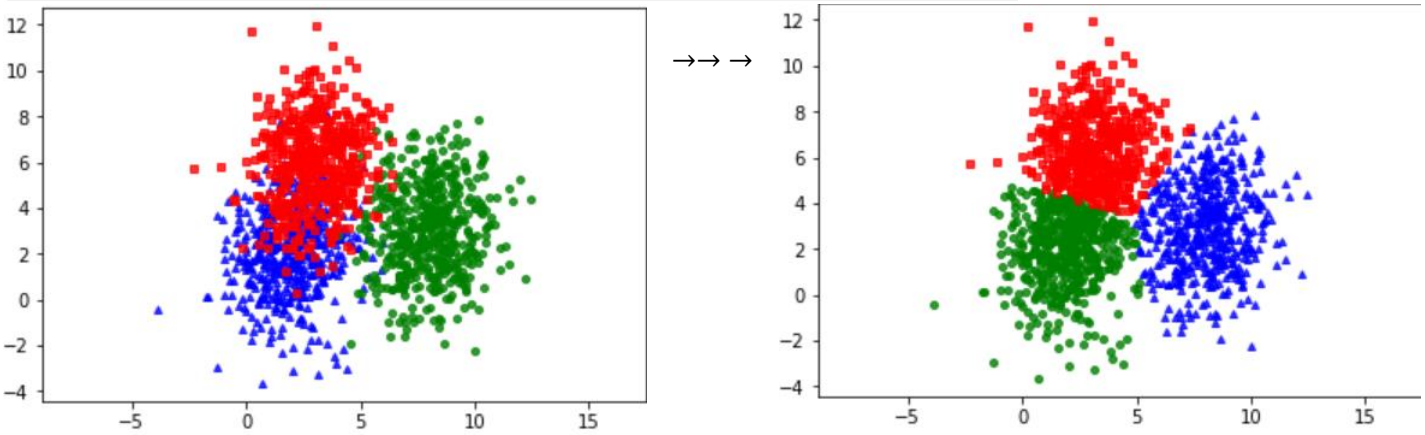


Hình 10.11: K-means clustering hoạt động không thực sự tốt trong trường hợp các cluster có số lượng phần tử chênh lệch hoặc các cluster không có dạng hình tròn (cầu).

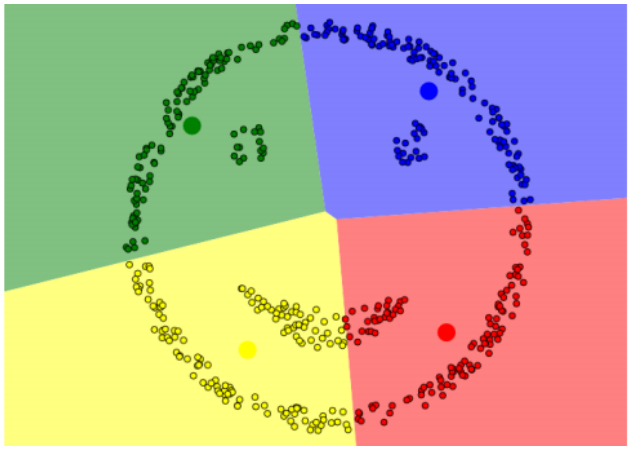
- The shape of cluster needs to be a circle (sphere): When the cluster follows normal distribution but the covariance matrix is not directly proportionally identity matrix, the clusters are not in circle (sphere) shape. KMeans clustering determines the label of data point based on the Euclid distance, so due to not circle shape, KMeans may not be effective. **Gaussian Mixture Model** may be able to solve the problem

```
means = [[2, 2], [8, 3], [3, 6]]
cov = np.array([[2, .5],[0, 3.5]], dtype=float)
N = 500
```

```
X0 = np.random.multivariate_normal(means[0], cov, N)
np.cov(X0.T)
```



- When one cluster is covered by another cluster: In the below image, intuitively, we separate into 4 clusters: left eye, right eye, mouse, other region (other region in face). However, because the left and right eye are in the face, so KMeans cannot give the correct clusters. In this case, **Spectral Clustering** may help



Hình 10.12: Một ví dụ về việc K-means clustering phân nhóm sai.

Application

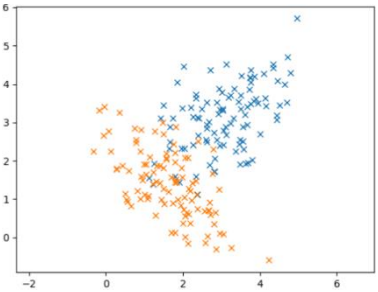
Despite the disadvantages mentioned before, KMeans is very necessary in ML and is the foundation of other complicated model

- The method, applied in Image Compression in which we replace the value of pixel by value of centroid, is called Vector Quantization (VQ). Moreover, VQ can be associated with Bag-of-Words
- VQ is also applied in searching the big data. Big data is clustered and the original value of each data point is assigned by its centroid. When any query is made, instead of calculating the distance between the query and data points, we just calculate the distance between the query and centroid, the return the data points have this centroid. Some keyword about this method: **Product Quantization, Cartesian k-means, Composite Quantization, Additive Quantization**

Extension

Gaussian Mixture Models

- Motivation



  - Because the data points overlap, so KMeans does not work well on it
  - Intuitively, it's clear that data points are generated from Gaussian because it has elliptical shape. Due to ellipse, not circle, KMeans does not care about covariance of the data (KMeans just works on covariance $\propto$ identity matrix)
  - Ellipse also shows the correlation between the features, e.g. the blue point seems to have relationship between X and Y. If 2 points that were equidistant from the center of the cluster, one followed this trend, one didn't, KMeans regards them equal, since it uses Euclidean distance
  - Practically, all dataset trends to normally distributed, **Central Limit Theorem**, when it has enough data points
  - Since we know these data are normally distributed, the idea of **GMM** is to find the parameters of the Gaussians that's most fit to our data
- Expectation – Maximization
  - Regard each point as being generated by *mixture of Gaussians* and compute:

$$\begin{cases} p(x) = \sum_{j=1}^{k} \phi_j N\left(x; \mu_j, \sum_j\right) \\ \sum_{j=1}^{k} \phi_j = 1 \end{cases}$$

    in which, j: number of cluster, ϕ_j : the weight of each Gaussian cluster, μ_j: the mean of Gaussian cluster j, $\sum_j$: covariance of each Gaussian cluster
  - Closed-form may be applied here, but ϕ_j need to be found, but if we know ϕ_j : combination of a Gaussians a particular point was taken from, but if ϕ_j can be found, $\mu_j, \sum_j$ can be easily calculated, so iterative method may be more appropriate: **EM**

- Expectation: rather than using hard assignment like in KMeans which, we compute the probability that each data point was generated by each of the *k* Gaussians or **soft assignment** by $W_j^{(i)} = \frac{\phi_j N\left(x; \mu_j, \sum_j\right)}{\sum_{q=1}^{k} \phi_q N\left(x; \mu_q, \sum_q\right)}$, we directly apply Gaussian and multiply it to ϕ_j, then normalize it to become the probability
- Maximization: our parameters like ϕ_j, $\mu_j, \sum_j$ are updated

$$\begin{cases} \phi_j = \frac{1}{N}\sum_{i=1}^{N} W_j^{(i)} \\ \mu_j = \frac{\sum_{i=1}^{N} W_j^{(i)} x^{(i)}}{\sum_{i=1}^{N} W_j^{(i)}} \\ \sum_j = \frac{\sum_{i=1}^{N} W_j^{(i)} \left(x^{(i)} - \mu_j\right)\left(x^{(i)} - \mu_j\right)^T}{\sum_{i=1}^{N} W_j^{(i)}} \end{cases}$$

- Summary
  - GMM works on arbitrary shape of data points but it should be non-overlap, if it overlap too much, the result is similar with KMeans