

## Clustering data into $k$ clusters

Given  $n$  points in  $\mathbb{R}^n$  split them into “similar” groups. This problem is tough even if the number of groups is known, which is the only case we consider.

**Input:**  $m$  feature vectors  $x_1, \dots, x_m$ , and the value of  $k$ .

**Output:** Partitioning of the data into  $k$  classes.

There are several ways to describe the partitions. One is an index function  $c(i)$ :

$$c(i) = j \quad \Leftrightarrow \quad x_i \text{ belongs to class } j$$

Another one is a membership function  $c(i, j)$ :

$$c(i, j) = \begin{cases} 1 & x_i \text{ belongs to class } j \\ 0 & \text{otherwise} \end{cases}$$

A probabilistic, or fuzzy version of the membership function is:

$$c(i, j) = p_{i,j}$$

subject to the conditions:  $p_{ij} \geq 0$ , and for each  $i$ ,  $\sum_j p_{ij} = 1$ . Here the value of  $p_{ij}$  is the probability, or the likelihood, that  $x_i$  belongs to class  $j$ .

### K-means clustering

The output of K-means is sometimes called *hard partition*. Define the quantization error by:

$$E = \sum_{j=1}^k \sum_{c(i)=j} |x_i - u_j|^2 = \sum_{i=1}^m |x_i - u_{c(i)}|^2$$

Keeping  $c(i)$  unchanged, the value of  $u_j$  that minimizes  $E$  is: can be computed by taking the derivative of  $E$  with respect to  $u_j$  and equating to 0. Observe that only the terms  $|x_i - u_j|^2$  depend on  $u_j$ , and

$$\frac{\partial}{\partial u_j} |x_i - u_j|^2 = \frac{\partial}{\partial u_j} (|x_i|^2 - 2x_i^T u_j + |u_j|^2) = 2u_j - 2x_i$$

Therefore:

$$\frac{\partial E}{\partial u_j} = \sum_{c(i)=j} (u_j - x_i) = m_j u_j - \sum_{c(i)=j} x_i$$

where  $m_j = \sum_{c(i)=j} 1$  is the number of points in class  $j$ . This gives:

$$\text{for all } j, \quad u_j = \frac{\sum_{c(i)=j} x_i}{m_j} \tag{1}$$

Keeping  $u_j$  unchanged, the value of  $c(i)$  that minimizes  $E$  is:

$$\text{for all } i, \quad c(i) = \arg \min_j |x_i - u_j|^2 \tag{2}$$

### Typical algorithm:

0. Start with a guess for  $u_1, \dots, u_k$ , or a guess for the  $c(i)$ .
1. Iterate until convergence the equations (1),(2).

It is easy to verify that each iteration reduced the error  $E$ , which proves convergence. Unfortunately the algorithm “almost always” gets stuck in a local minimum. The standard approach is to run it many times with different initial estimates, and select the partitioning that gives the smallest value of  $E$ .

### Lloyd’s algorithm

Lloyd described a specific formulation of randomizing the initial step of the algorithm. In Lloyd’s version Step 0 is implemented as follows:

0. Select the initial  $u_i$  uniformly at random from  $x_1, \dots, x_m$ .

### Kmeans++

Here we spend more time on the initial step, the initial selection of  $u_1, \dots, u_k$ .

- 0.1. Select  $u_1$  uniformly at random from  $x_1, \dots, x_m$ .
- 0.2. Assuming that  $u_1, \dots, u_t$  are already selected, with  $t < k$ . Use the following procedure to select  $u_{t+1}$ :

- 0.2.1 for each  $x_i$  compute the value  $d_i$  as:

$$d_i = \min_j |x_i - u_j|^2$$

- 0.2.2 Compute the probability values:

$$Z = \sum_i d_i, \quad p_i = \frac{d_i}{Z}$$

- 0.2.3 Select one of the  $x_i$  at random according to the probabilities  $p_i$ . Set it as  $u_{t+1}$ .