Clustering

Clustering is a member of unsupervised learning. It could be divided into three categories roughly: prototype-based clustering, density-based clustering and hierarchal clustering.

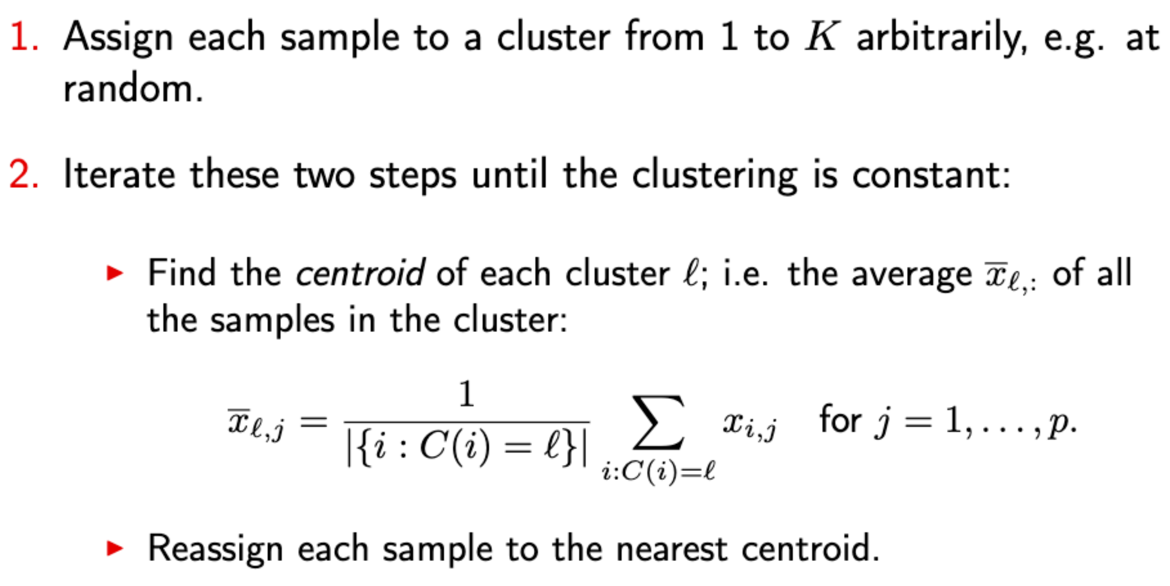
- Prototype-based clustering: we expect that clustering structure could be described by a set of prototypes. (K-means, K-medoids, Learning vector quantization, Gaussian mixture model)
- Density-based clustering: we expect that clustering structure could be described by the density of samples. (DBSCAN)
- Hierarchical clustering: tree clustering structure.

K-Means

1. Algorithm

The goal of this method is to maximize the similarity of samples within each cluster:

$$W(C) = \frac{1}{2}\Sigma_{k=1}^{K}\Sigma_{C(i)=k}\Sigma_{C(j)=k}d(x_i, x_j)$$



1. Assign each sample to a cluster from 1 to K arbitrarily, e.g. at random.
2. Iterate these two steps until the clustering is constant:
   - Find the *centroid* of each cluster ℓ; i.e. the average $\overline{x}_{\ell,:}$ of all the samples in the cluster:

     $$\overline{x}_{\ell,j} = \frac{1}{|\{i : C(i) = \ell\}|} \sum_{i:C(i)=\ell} x_{i,j} \quad \text{for } j = 1, \dots, p.$$

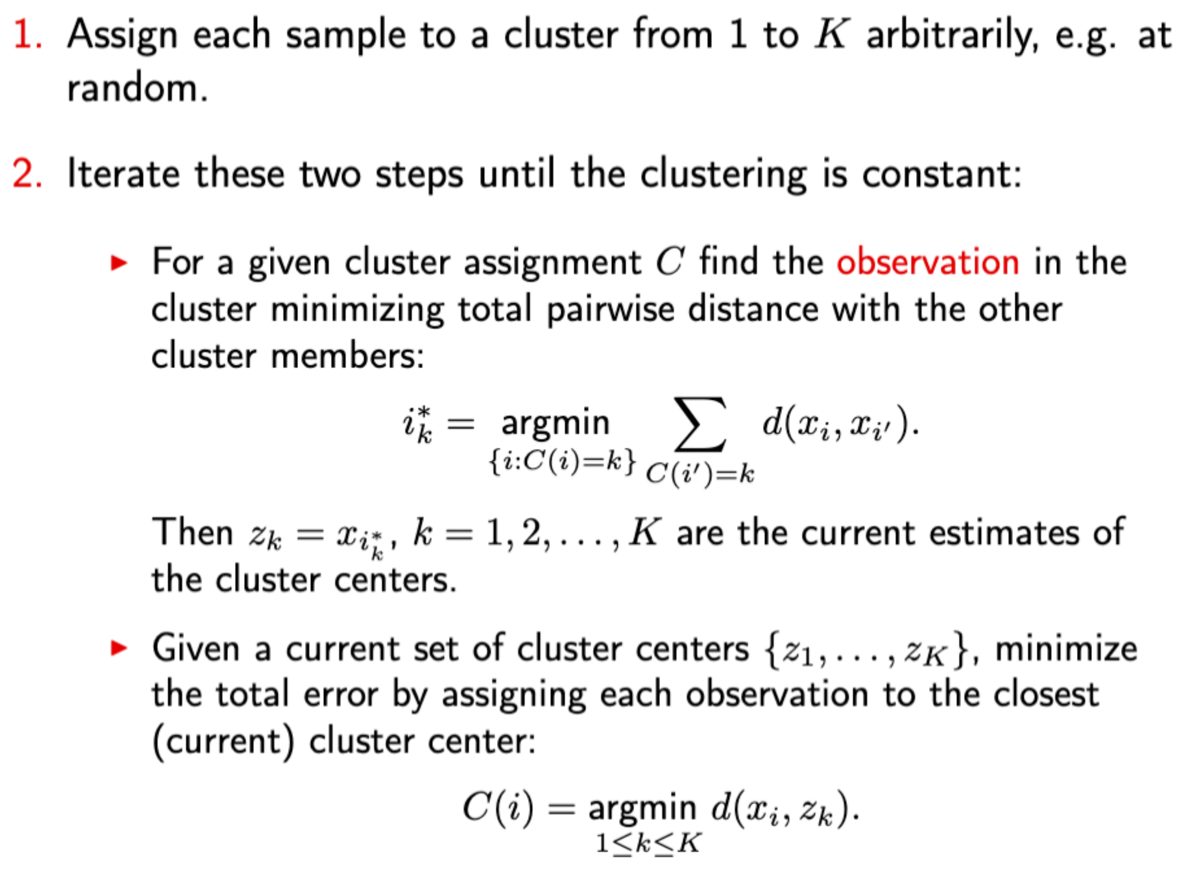
   - Reassign each sample to the nearest centroid.

2. Additionals

- Different initialization could yield a different result.
- In practise, we usually start from many random initializations and choose the one which minimizes the objective function.
- K could be tuned via cross validation.
- Euclidean distance is not the only choice for k-means, sometimes we could also try to use correlation distance.

K-Medoids

1. Algorithm



1. Assign each sample to a cluster from 1 to K arbitrarily, e.g. at random.
2. Iterate these two steps until the clustering is constant:
   - For a given cluster assignment C find the observation in the cluster minimizing total pairwise distance with the other cluster members:
   $$i_k^* = \operatorname*{argmin}_{\{i:C(i)=k\}} \sum_{C(i')=k} d(x_i, x_{i'}).$$
   Then $z_k = x_{i_k^*}$, $k = 1, 2, \ldots, K$ are the current estimates of the cluster centers.
   - Given a current set of cluster centers $\{z_1, \ldots, z_K\}$, minimize the total error by assigning each observation to the closest (current) cluster center:
   $$C(i) = \operatorname*{argmin}_{1\le k\le K} d(x_i, z_k).$$

2. Additionals

Comparing with k-means, k-medoids clustering is more explanable because the centroid is required to be one of the observations. And one only needs pairwise distances for K-medoids rather than the raw observations.

Learning Vector Quantization

The core idea of LVQ is that although we do not know the labels we want, we could get some lights from other labels we know. LVQ is usually used to make the clustering in details.

1. Algorithm

- Initialize a set of prototype vectors $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_K}$ and its labels $\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_K}$
- For each iteration
  - Choose one sample randomly $\mathbf{x_j}, y_j$ and then find its closest prototype vector $\mathbf{p_{i*}}$.
  - If $y_i = t_{i*}$ then
  $$p' = p_{i*} + \eta\,(\,x_j - p_{i*}\,)$$
  else

$$p' = p_{i*} - \eta (x_j - p_{i*})$$

- Finally, we would get a set of prototype vectors.

DBSCAN

The main concept of DBSCAN (density-based spatial clustering of applications with noise) is to locate regions of high density that are separeated from one another by regions of low density.

1. Algorithm

```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                 /* Cluster counter
*/
    for each point P in database DB {
        if label(P) ≠ undefined then continue             /* Previously
processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)    /* Find neighbors
*/
        if |N| < minPts then {                            /* Density check */
            label(P) = Noise                              /* Label as Noise
*/
            continue
        }
        C = C + 1                                         /* next cluster
label */
        label(P) = C                                      /* Label initial
point */
        Seed set S = N \ {P}                              /* Neighbors to
expand */
        for each point Q in S {                           /* Process every
seed point */
            if label(Q) = Noise then label(Q) = C         /* Change Noise to
border point */
            if label(Q) ≠ undefined then continue         /* Previously
processed */
            label(Q) = C                                  /* Label neighbor
*/
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /* Find neighbors
*/
            if |N| ≥ minPts then {                        /* Density check */
                S = S ∪ N                                 /* Add new
neighbors to seed set */
            }
        }
    }
}
```

2. Additionals

- DBSCAN is great at separating clusters of high density versus clusters of low density within a given dataset. And it is good at clustering dataset distributed in any shapes.
- However, it would struggle with high dimensionality data because most of points could be marked as noise due to the sparsity in high dimensions.

Ref: https://towardsdatascience.com/dbscan-algorithm-complete-guide-and-application-with-python-scikit-learn-d690cbae4c5d & https://blog.csdn.net/xc_zhou/article/details/88316299

Hierarchical Clustering

The core idea of hierarchical clustering is to keep merging two clusters till we get the number of clusters we want.

1. Algorithm

- At first, we take each sample as one cluster.
- Compute distance matrix for every pair of samples.
- Repeat:

  Merge the two closest clusters and update the distance matrix
- Until only a single cluster remains.

2. Additionals

- Distance Metrics

  There are three methods to compute distance matrix for hierarchical clustering.
  - Minimal distance (single-linkage): the distance between the closest samples in two clusters.
  - Maximum distance (complete-linkage): the distance between the farest samples in two clusters.
  - Average distance (average-linkage): the average distance between all samples in two clusters.

Gaussian Mixture Model

In gaussian mixture model, we expect that samples could be clustered by probablic models. Mixture model could be represented as

$$\pi(x) = \Sigma_{k=1}^{K} c_k p(x|\theta_k)$$

For GMM, $p(x|\theta_k)$ is gaussian distribution and $\Sigma_{k=1}^{K} c_k = 1$. Here we would solve the problem via EM algorithm.

1. EM Algorithm

- Why should be EM algorithm?

  For ideal optimization problem, we could easily get closed-form solution by taking deriatives. But often we cannot find such a solution for θ . In this case, we would also use gradient descent to **approximate** the solution. However, sometimes we do want to write out the solution exactly rather than approximation. How can we do that? The answer is EM algorithm.

- Setting for EM algorithm
  - What we want: maximize the posterior probability

$$\begin{aligned}\theta &= argmax_\theta ln(\theta|X) \\ &\approx argmax_\theta ln(X,\theta)\end{aligned}$$

  - What we have:

$$\begin{aligned}&X \sim p(X|\theta) \\ &\theta \sim p(\theta)\end{aligned}$$

  EM starts with an assumption: Assume there is some other variable ϕ that we can *introduce* to the model so that the marginal distribution is unchanged. That is

$$p(X,\theta) = \int p(X,\theta,\phi)d\phi$$

  Firstly, according to bayes rule, we could know that

$$\begin{aligned}&lnp(X,\theta) = lnp(X,\theta,\phi) - lnp(\phi|X,\theta) \\ &q(\phi)lnp(X,\theta) = q(\phi)lnp(X,\theta,\phi) - q(\phi)lnp(\phi|X,\theta) \\ &\int q(\phi)lnp(X,\theta)d\phi = \int q(\phi)lnp(X,\theta,\phi)d\phi - \int q(\phi)lnp(\phi|X,\theta)d\phi\end{aligned}$$

  Then, we have

$$\begin{aligned}lnp(X,\theta) &= \int q(\phi)lnp(X,\theta,\phi)d\phi - \int q(\phi)lnq(\phi)d\phi - \int q(\phi)lnp(\phi|X,\theta)d\phi + \int q(\phi)lnq(\phi)d\phi \\ &= \int q(\phi)ln\frac{p(X,\theta,\phi)}{q(\phi)}d\phi + \int q(\phi)ln\frac{q(\phi)}{p(\phi|X,\theta)}d\phi \\ &= L(\theta) + KL(q||p)\end{aligned}$$

  In fact, $KL(q||p) \geq 0$.

  - How the algorithm works?
    - E-step: set $q_t(\phi) = p(\phi|X,\theta_{t-1})$ and calculate

      $L_t(\theta) = \int q_t(\phi)lnp(X,\theta,\phi)d\phi - \int q_t(\phi)lnq_t(\phi)d\phi$
    - M-step: treat $L_t(\theta)$ as a function of θ and find the value of θ that maximizes it. Actually, in practice it is only necessary to calculate $\int q_t(\phi)lnp(X,\theta,\phi)d\phi$ and maximize this over θ
    - Note, if we cannot solve $\nabla_\theta L_t(\theta) = 0$ analytically which means you have to use gradient methods to optimize it, you might as well just use gradient methods to optimize $lnp(X,\theta)$ instead.
    - In EM algorithm, it could be proved that θ after each iteraction would make our objective function $\ln p(X,\theta)$ **monotonically increasing**.

2. GMM with EM

- Initialize parameters of GMM $\alpha_i, \mu_i, \Sigma_i$

- For each sample ($j = 1, 2, \ldots, m$), calculate the posterior probability $\gamma_{ji} = \frac{\alpha_i p(x_j|\mu_i, \Sigma_i)}{\Sigma_{l=1}^{k} \alpha_l p(x_j|\mu_l, \Sigma_l)}$
- For each class ($i = 1, 2, \ldots, k$), update parameters
  - $\mu'_i = \frac{\Sigma_{j=1}^{m} \gamma_{ji} x_j}{\Sigma_{j=1}^{m} \gamma_{ji}}$
  - $\Sigma'_i = \frac{\Sigma_{j=1}^{m} \gamma_{ji} (x_j - \mu'_i)(x_j - \mu'_i)^T}{\Sigma_{j=1}^{m} \gamma_{ji}}$
  - $\alpha'_i = \frac{\Sigma_{j=1}^{m} \gamma_{ji}}{m}$
- Stop iteration till the parameters converge.