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)
- Hierarchal 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:

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

   - 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:

     $$i_k^* = \underset{\{i:C(i)=k\}}{\text{argmin}} \sum_{C(i')=k} d(x_i, x_{i'}).$$

     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:

     $$C(i) = \underset{1 \le k \le K}{\text{argmin}}\ d(x_i, z_k).$$

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