1 Subjects

- Representation based
- Density based
- Subspace

2 Notes

2.1 Representation based

Here are some of the main algorithms within representation based clustering:

- K-Means
- K-Medoids

Any more?

Common for all of these algorithms, is that they all use some point which summarizes or "represents" the cluster, a common choice being the mean (or the *centroid*) μ_i. If we were to make this partitioning in an exhaustive way, we would simply find all possible partitions (i.e. all possible means) and pick the best. Doing this brute-force approach, results in $\mathcal{O}\left(k^n/k!\right)$ clusterings of n points into k groups, so this is not practically feasible at all. Let's instead look at some smarter algorithms for solving this problem.

2.1.1 K-Means

The idea of k-means is for some k to form k groups so that the sum of the (squared) distances between the mean of the groups and their elements is minimal. In other words we want to minimize the squared error measure as we often do for linear regression.

Given that our points $p_i = (p_{i1}, \dots, p_{id})$ are a point in a d-dimensional vector space, where the mean of a set of points is defined (e.g. euclidean). We then have that the centroid μ_C for some cluster C is defined as:

$$\mu_C = \frac{1}{|C|} \sum_{p_i \in C} p_i$$

We can then compute the *sum of squared errors* as:

$$SSE(C) = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

We then want to find the clustering that minimizes SSE:

$$C^* = \arg\min_C SSE(C)$$

We can outline the k-means algorithm as:

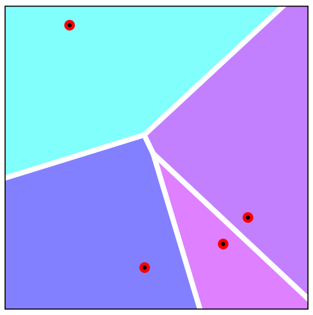
1. Partition the objects into k non-empty subsets
2. Compute the centroids of the clusters of the current partition. The centroid is the center (mean point) of the cluster.
3. Assign each object to the cluster with the nearest representative.
4. Go back to Step 2, stop when representatives do not change.

Which gives us the following pseudocode: Then, if we are using euclidean dis-

Algorithm 1 k-means

```
procedure k-MEANS($D, k, \epsilon$)                                ▷ asd
    $t \leftarrow 0$
    Randomly initialize $k$ centroids $\mu_{t1}, \mu_{t2}, \ldots, \mu_{tk} \in \mathbb{R}^d$
    repeat
        $t \leftarrow t + 1$
        $C_j \leftarrow \emptyset$ for all $j = 1, \ldots, k$
        // Cluster assignment step
        for all $x_j \in D$ do
            $j^* \leftarrow \arg\min_{i=1,\ldots,k} \|x_j - \mu_{ti}\|^2$          ▷ Assign $x_j$ to closest centroid
            $C_{j^*} \leftarrow C_{j^*} \cup \{x_j\}$
        end for
        // Centroid Update Step
        for all $i = 1$ to $k$ do
            $\mu_{ti} \leftarrow \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j$
        end for
    until $\sum_{i=1}^{k} \|\mu_{ti} - \mu_{t-1,i}\|^2 \leq \epsilon$
end procedure
```

tance (the manhattan distance could look differently), the voronoi diagram (a visual representation of the clusters) will look like this image:



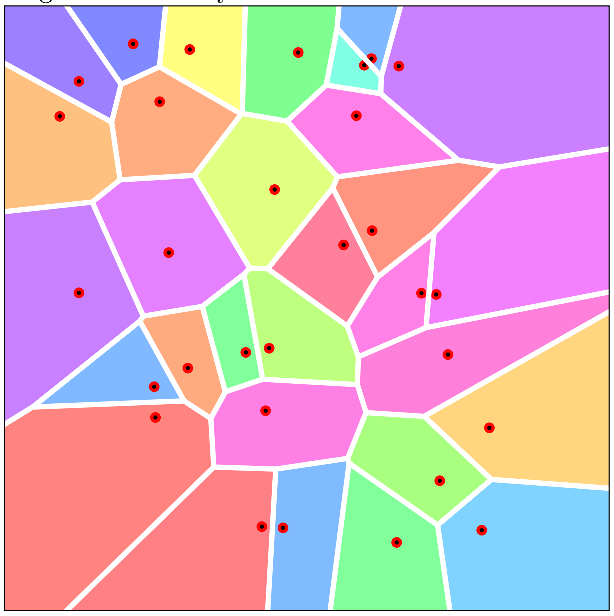
Strengths

- Relatively efficient $\mathcal{O}(tkn)$ where n is #objects, k is #clusters and t is #iterations

- Normally k and t are both much smaller than n so in practice it's only n that matters.
- It's easy to implement

Weaknesses

- Applicable only in vector spaces where the mean is defined
- We need to specify the number of clusters k in advanced (it's a hyper-parameter)
- It's sensitive to noisy data and outliers
- Clusters are forced to have convex shapes, for example here is a voronois diagram with many clusters:



- Both results and running time are very dependent on the initial selection of k-means, as it often terminates at a *local optimum*, however there do exist methods for good initialization.

There are several variant of k-means, e.g. ISODATA which extends k-means by merging and splitting clusters to eliminate very small clusters, at a cost of more hyper-parameters.

2.1.2 K-Medoids

k-means implicitly assume euclidean distance, since we need to minimize the distance to the mean, there are variations for other distance functions.

The k-medoid algorithm is more general, it's motivated by the L_1 norm (Manhattan) distance and also works in spaces where a mean is not defined. We just need to be able to compute pairwise distances. Furthermore k-medoid will turn out to be more robust to noise. The general L_p distance metric (Minkowski-Distance) is defined as:

$$d_p(x,y) = \sqrt[p]{\sum_{i=1}^{d} |x_i - y_i|^p}$$

Where Euclidean is the L_2 metric and Manhattan is the L_1 metric. Other examples of distance functions is the maximum metric:

$$d_\infty(x,y) = \max_{1 \le i \le d} |x_i - y_i|$$

Or for two sets we could define it as:

$$d_{set}(x,y) = \frac{|x \cup y| - |x \cap y|}{|x \cup y|}$$

Or we could use the Hamming distance etc.

Now for the basic idea of k-medoid, we start by defining some notions. First of all, the medoid m_C is the representative object for a cluster C. The compactness of a clustering C is measured as:

$$TD = \sum_{i=1}^{k} \sum_{p \in C} \text{dist}(p, m_C)$$

We can then comput the k-medoids, using PAM as follows:

1. Select k objects arbitrarily as medoids, then assign each remaining (non-medoid) object to the cluster with the nearest representative. We will then compute the TD and name it $TD_{current}$
2. For each pair (medoid M, non-medoid N), exhaustively compute the TD value for the partition if N was the medoid instead of M, TD_{NM}.
3. Select the non-medoid N for which the TD_{NM} value was minimal, if the TD value is smaller than $TD_{current}$ then:
   (a) Swap N with M
   (b) Set $TD_{current} \leftarrow TD_{NM}$
   (c) Go back to Step 2
4. Stop

Strength

- We can use it on arbitrary objects (e.g. points or sets) and arbitrary distance measures
- Not quite as sensitive to noisy data and outliers as k-means

Weaknesses

- Inefficient
- We need to specify number of clusters k in advance and clusters need to have convex shapes

2.1.3 Expectation Maximization (EM)

In k-means, each point could only belong to precisely one cluster, EM is soft-assignment of points to clusters, so each point has a probability of belonging to each cluster. We represent a cluster by a probability distribution, usually we will represent it as center point μ_C and a $d \times d$ covariance matrix Σ_C for the points in the cluster C. If we assume that the each cluster C_i is characterized by a multivariate normal distribution, then we can define the density function for cluster C as:

$$P(x|C) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_C|}} \cdot e^{-\frac{1}{2}(x-\mu_C)^T \cdot (\Sigma_C)^{-1} \cdot (x-\mu_C)}$$

We can then estimate the a-priori probability of class C_i, $P(C_i)$ as the relative frequency W_i. I.e. we define the $P(C_i)$ as the fraction of contribution from the entire data-set D to class C_i:

$$P(x) = \sum_{i=1}^{k} W_i \cdot P(x|C_i)$$

We can then compute the probability that x belongs to cluster C_i as:

$$P(C_i|x) = W_i \cdot \frac{P(x|C_i)}{P(x)}$$

Now, we can compute a measure of the quality of a clustering M as $E(M)$ which indicates the probability that the data D has been generated by the distribution model M:

$$E(M) = \sum_{x \in D} \log(P(x))$$

Which is what we will try to maximize in EM. Which brings us to our algorithm:

During the maximization step we will need to compute the following values: The weight W_i of cluster C_i which is the estimate for the previous probability of each cluster. The center μ_i of cluster C_i and the covariance matrix Σ_i of cluster C_i.

Algorithm 2 Expectation Maximization

```
procedure EM($D, k$)
    Generate an initial model $M' = (C'_1, \ldots, C'_k)$
    repeat
        // (Re-) assign points to clusters - expectation step
        for all $x \in D$ do
            for all $C_i$, $i = 1, \ldots, k$ do
                Compute $P(x|C_i)$, $P(x)$ and $P(C_i|x)$
            end for
        end for
        // (Re-) compute the model - maximization step
        for all $C_i$, $i = 1, \ldots, k$ do
            Recompute $W_i$, $\mu_C$ and $\Sigma_C$
            Compute a new model $M = \{C_1, \ldots, C_k\}$ using $W_i$, $\mu_C$ and $\Sigma_C$
            Replace $M'$ with $M$
        end for
    until $|E(M) - E(M')| < \epsilon$
end procedure
```

We can estimate the weight as the fraction of weights that contribute to the cluster:

$$W_i = \frac{1}{|D|} \sum_{x \in D} P(C_i|x)$$

We can estimate the mean as the weighted average of all points:

$$\mu_i = \frac{\sum_{x \in D} x \cdot P(C_i|x)}{\sum_{x \in D} P(C_i|x)}$$

And lastly we can re-estimate the covariance matrix as the weighted covariance over all combinations of dimensions:

$$\Sigma_i = \frac{\sum_{x \in D} P(C_i|x)(x - \mu_i)(x - \mu_i)^T}{\sum_{x \in D} P(C_i|x)}$$

Currently the covariance matrix is a $d \times d$ matrix which can be a quite costly as we have to estimate d^2 parameters and often we don't have enough data for a reliable estimation. An optimization we could make here would be to assume that all dimensions are independent and only estimate the d parameters that make up the diagonal of the matrix.

Strengths

- We converge to a minimum (which might be local)
- Rather efficient $\mathcal{O}(n \cdot k \cdot \#iterations)$
- However #iterations is quite high in many cases unlike k-means

Weaknesses

- Both result and runtime depends highly on the initial assignment
- Also depends strongly on a proper choice of parameter k

Furthermore, with EM, objects may belong to several clusters. If we want it to be hard-assignment, then we can assign each object to the cluster which it has highest probability of belonging to.

A last note, a good initialization of EM, is often to run k-means first to return a crude estimate of the means and the finetune the clustering with EM.

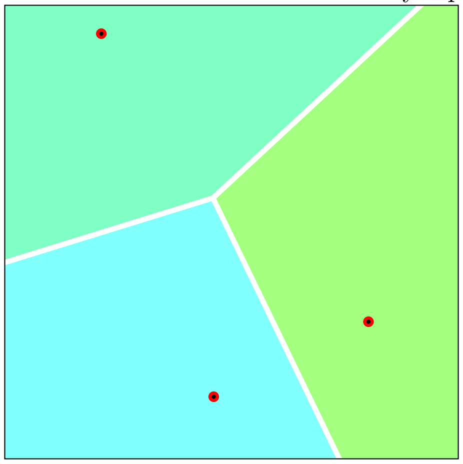
2.1.4 Initialization of representative based clustering

Here is an approach sugested by [Fayyad, Reina and Bradley 1998]:

- Draw M different (small) samples of the dataset $S_1, S_2, \ldots, S_M$
- Cluster each sample, such that we get M estimates for k representatives:
$$S_i = (S_{i1}, S_{i2}, \ldots, S_{ik})$$
- We then cluster the unioned set: $DS = S_1 \cup S_2 \cup \cdots \cup S_M$, one time for each of our M estimates for the k representatives, leaving us with M different clusterings of DS.
- Now use the best of these M clusterings as initialization for the partitioning clustering of the whole data-set.

2.2 Density based clustering

Remember how k-means worked by seperating into a voronoid diagram?



In general representation-based clustering only work for convex clusters, Density-based clustering attempts to cluster more complex shapes, like a S-shaped cluster. The basic concept is as follows:

- For any point in a cluster, the local point density around that point has to exceed some threshold
- The set of points from one cluster is spatially connected

Local point density at a point p is defined by two parameters:

- ϵ - radius for the neighbourhood of point q:

$$N_\epsilon(q) = \{p \text{ in data-set } D | \text{dist}(p, q) \leq \epsilon\}$$

- MinPts - minimum number of points in the given neighbourhood $N(p)$

We call an object q a core object (or core point) w.r.t. ϵ if $|N_\epsilon(q)| \geq MinPts$ i.e. if there are at least $MinPts$ objects within a radius of ϵ of q.

We now define the following terms:

- p is ***directly density-reachable*** from q w.r.t. ϵ and $MinPts$ if:
  1. $p \in N_\epsilon(q)$
  2. q is a core object w.r.t. ϵ and $MinPts$
- ***density-reachable*** is a transitive closure of *directly* density-reachable
- p is ***density-connected*** to a point q w.r.t. ϵ and $MinPts$, if there is a point o such that both p and q are density-reachable from o w.r.t. ϵ and $MinPts$.
- A ***density-based cluster*** is a non-empty subset of S of dataset D satisfying:
  1. Maximality: if p is in S and q is density-reachable from p then q is in S
  2. Connectivity: each object in S is density-connected to all other objects
- A ***density-based clustering*** of a dataset is $D : \{S_1, \dots, S_n; N\}$ where:
  - $S_1, \dots, S_n$ is all density-based clusters in the data-set D
  - $N = D \setminus \{S_1, \dots, S_n\}$ is called the ***noise*** (the objects that are not in any cluster)

One clarification, imagine two non-core points p and q with a core point c between them. Since neither p nor q are core points, they cannot be density-reachable, but they can be density-connected because of c.

2.3 Measuring the quality of a clustering

For choosing k, one idea could be to determine the clustering for $k = 2, 3, \dots, n-1$ and then choose the "best" clustering. But how do we actually measure the quality of a clustering? If we need to use it to select the best k, it has to be independent of k, and the measures for compactness of a clustering (TD) are decreasing with increasing values of k.

So we define the silhouette coefficient, with the basic idea that:

- Quality of clustering = how appropriate is the mapping of objects to clusters.
- Elements in a cluster should be "similar" to their representative, so we should measure the distance of objects to their representative a.
- Elements in different clusters should be different, so we measure the average distance of objects to an alternative cluster (second closest) b.

We can define $a(o)$ as the average distance between object o and the objects in its cluster A:

$$a(o) = \frac{1}{|C(o)|} \sum_{p \in C(o)} \mathrm{dist}(o, p)$$

And we can define $b(o)$ as the average distance between object o in its "second closest" cluster B

$$b(o) = \min_{C_i \neq C(o)} \frac{1}{|C_i|} \sum_{p \in C_i} \mathrm{dist}(o, p)$$

Then we can compute the silhouette of o as:

$$s(o) = \frac{b(o) - a(o)}{\max\{a(o), b(o)\}}$$

The values of the silhouette coefficient range from -1 to $+1$ and we can read the values as:

- $s(o) = -1$: bad, on average o is closer to members of B.
- $s(o) = 0$: o is somewhere in-between A and B
- $s(o) = 1$: good, o is on average closest to its cluster A

The silhouette coefficient s_C of a clustering is simply the average silhouette of all objects and we can read the value as:

- $0.7 < s_C \leq 1.0$ strong structure
- $0.5 < s_C \leq 0.7$ medium structure
- $0.25 < s_C \leq 0.5$ weak structure
- $s_C \leq 0.25$ no structure