8 Clustering

This topic will focus on automatically grouping data points into subsets of similar points. There are numerous ways to define this problem, and most of them are quite messy. And many techniques for clustering actually lack a mathematical formulation. We will focus on what is probably the cleanest and most used formulation: k-means clustering. But, for background, we will begin with a mathematical detour in Voronoi diagrams.

8.1 Voronoi Diagrams

Consider a set $S = \{s_1, s_2, \ldots, s_k\} \subset \mathbb{R}^d$ of sites. We would like to understand how these points carve up the space $\mathbb{R}^d$.

We can think of this more formally as the *post office problem*. Let these sites define the locations of a post office. For all points in $\mathbb{R}^d$ (e.g., a point on the map for points in $\mathbb{R}^2$), we would like to assign it to the closest post office. For a fixed point, we can just check the distance to each post office:

$$\phi_S(x) = \arg\min_{s_i \in S} \|x - s_i\|.$$

However, this may be slow (naively take $O(k)$ time for each point x), and does not provide a general representation or understanding for all points. The "correct" solution to this problem is the Voronoi diagram.

The Voronoi diagram decomposes $\mathbb{R}^d$ into k regions (a *Voronoi cell*), one for each site. The region for site s_i is defined.

$$R_i = \{x \in \mathbb{R}^d \mid \phi_S(x) = s_i\}$$

If we have these regions nicely defined, this solves the post office problem. For any point x, we just need to determine which region it lies in (for instance in $\mathbb{R}^2$, once we have defined these regions, through an extension of binary search, we can locate the region containing any $x \in \mathbb{R}^2$ in only $O(\log k)$ time). But what do these regions look like, and what properties do they have.

Voronoi edges and vertices. We will start our discussion in $\mathbb{R}^2$. Further, we will assume that the sites S are in *general position*: in this setting, it means that no set of three points lie on a common line, and that no set of four points lie on a common circle.

The boundary between two regions R_i and R_j, called a *Voronoi edge*, is a line or line segment. This edge $e_{i,j}$ is defined

$$e_{i,j} = \{x \in \mathbb{R}^2 \mid \|x - s_i\| = \|x - s_j\| \leq \|x - s_\ell\| \text{ for all } \ell \neq i, j\}$$

as the set of all points equal distance to s_i and s_j, and not closer to any other point s_ℓ.

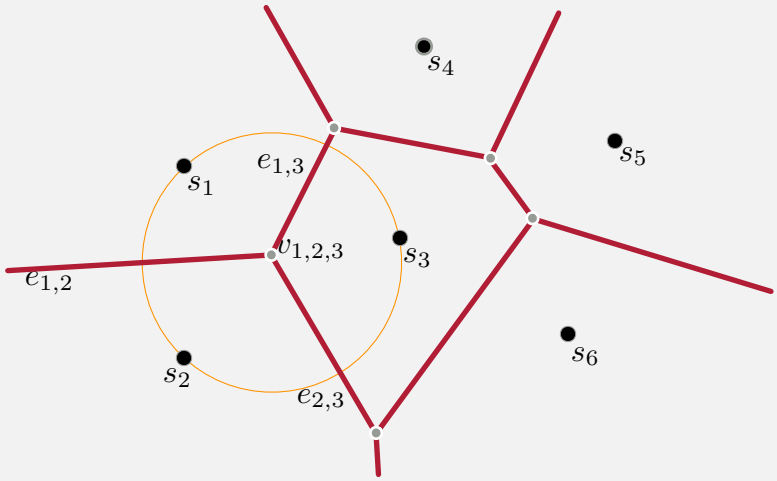
Why is this set a line segment? If we only have two points in S, then it is the bisector between them. Draw a circle centered at any point x on this bisector, and if it intersects one of s_i or s_j, it will also intersect the other. This is true since we can decompose the squared distance from x to s_i along orthogonal components: along the edge, and perpendicular to the edge from s_i to $\pi_{e_{i,j}}(s_i)$.

Similarly, a Voronoi vertex $v_{i,j,\ell}$ is a point where three sites s_i, s_j, and s_ℓ are all equidistance, and no other points are closer:

$$v_{i,j,k} = \{x \in \mathbb{R}^2 \mid \|x - s_i\| = \|x - s_j\| = \|x = s_\ell\| \leq \|x - s_k\| \text{ for all } k \neq i, j, \ell\}.$$

This vertex is the intersection (and end point) of three Voronoi edges $e_{i,i}$, $e_{i,\ell}$, and $e_{j,\ell}$. Think of sliding a point x along an edge $e_{i,j}$ and maintaining the circle centered at x and touching s_i and s_j. When this circle grows to where it also touches s_ℓ, then $e_{i,j}$ stops.

Example: Voronoi Diagram
See the following example with $k = 6$ sites in $\mathbb{R}^2$. Notice the following properties: edges may be unbounded, and the same with regions. The circle centered at $v_{1,2,3}$ passes through s_1, s_2, and s_3. Also, Voronoi cell R_3 has $5 = k - 1$ vertices and edges.



Size complexity. So how complicated can these Voronoi diagrams get? A single Voronoi cell can have $k-1$ vertices and edges. So can the entire complex be of size $O(k^2)$ (each of k regions requiring complexity $O(k)$)? No. The Voronoi vertices and edges describe a planar graph. Some cool results from graph theory says that planar graphs have asymptotically the same number of edges, faces, and vertices. Euler's Formula for a planar graph with n vertices, m edges, and k faces is that $k + n - m = 2$. And Kuratowski's criteria says for $n \geq 3$, then $m \leq 3n - 6$. Hence, $k \leq 2n - 4$ for $n \geq 3$. The duality construction to Delauney triangulations (discussed below) will complete the argument. Since there are k faces (the k Voronoi cells, one for each site), then there are also $O(k)$ edges and $O(k)$ vertices. In particular, there will be precisely $2n - 5$ vertices and $3k - 6$ edges.

However, this does not hold in $\mathbb{R}^3$. In particular, for $\mathbb{R}^3$ and $\mathbb{R}^4$, the complexity (number of cells, vertices, edges, faces, etc) is $O(k^2)$. This means, there could be roughly as many edges as their are pairs of vertices!

But it can get much worse. In $\mathbb{R}^d$ (for general d) then the complexity is $O(k^{\lceil d/2 \rceil})$. This is a lot. Hence, this structure is impractical to construct in high dimensions.

: *The **curse** of dimensionality! ooooh*

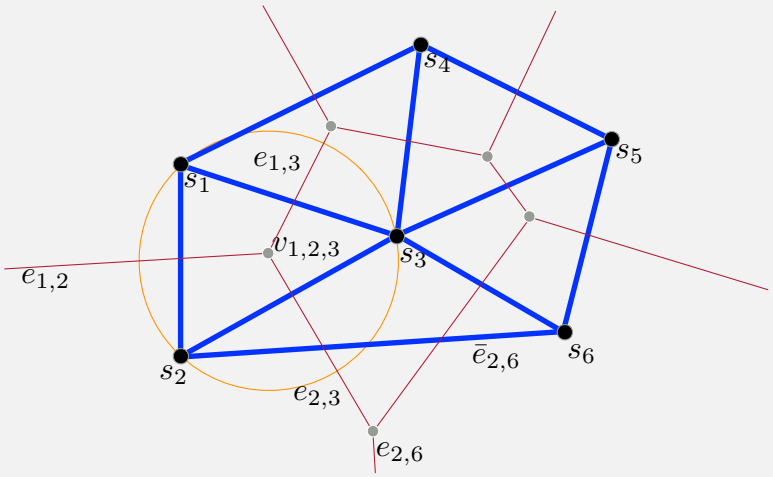
Moreover, since this structure is explicitly tied to the post office problem, and the nearest neighbor function ϕ_S, it indicates that (a) in $\mathbb{R}^2$ this function is nicely behaved, but (b) in high dimensions, it is quite complicated.

8.1.1 Delaunay Triangulation

A fascinating aspect of the Voronoi diagram is that it can be converted into a very special graph where the sites S are vertices, one called the *Delaunay triangulation*. This is the *dual* of the Voronoi diagram.

- Each face R_i of the Voronoi diagram maps to a vertex s_i in the Delaunay triangulation.
- Each vertex $v_{i,j,\ell}$ in the Voronoi diagram maps to a triangular face $f_{i,j,\ell}$ in the Delaunay triangulation.
- Each edge $e_{i,j}$ in the Voronoi diagram maps to an edge $\bar{e}_{i,j}$ in the Delaunay triangulation.

Example: Delaunay Triangulation
See the following example with 6 sites in $\mathbb{R}^2$. Notice that every edge, face, and vertex in the Delaunay triangulation corresponds to a edge, vertex, and face in the Voronoi diagram. Interestingly, the associated edges may not intersection; see $e_{2,6}$ and $\bar{e}_{2,6}$.



Because of the duality between the Voronoi diagram and the Delaunay triangulation, their complexities are the same. That means the Voronoi diagram is of size $O(k)$ for k sites in $\mathbb{R}^2$, but more generally is of size $O(k^{\lceil d/2 \rceil})$ in $\mathbb{R}^d$.

The existence of of the Delaunay triangulation shows that there always exist a triangulation: A graph with vertices of a given set of points $S \subset \mathbb{R}^2$ so that all edges are straightline segments between the vertices, and each face is a triangle. In fact, there are many possible triangulations: one can always simply construct some triangulation greedily, draw any possible edges that does not cross other edges until no more can be drawn.

The Delaunay triangulation, however, is quite special. This is the triangulation that maximizes the smallest angle over all triangles; for meshing applications in graphics and simulation, skinny triangles are very hard to deal with, so this is very useful.

In circle property. Another cool way to define the Delaunay triangulation is through the *in circle* property. For any three points, the smallest enclosing ball either has all three points on the boundary, or has two points on the boundary and they are antipodal to each other. Any circle with two points antipodal on the boundary s_i and s_j, and contains no other points, then the edge $e_{i,j}$ is in the Delaunay triangulation. This is a subset of the Delaunay triangulation called the *Gabriel graph*.

Any circle with three points on its boundary s_i, s_j, and s_ℓ, and no points in its interior, then the face $f_{i,j,\ell}$ is in the Delaunay triangulation, as well as its three edges $e_{i,j}$, $e_{i,\ell}$ and $e_{j,\ell}$. But does not imply those edges satisfy the Gabriel property.

For instance, on a quick inspection, (in the example above) it may not be clear if edge $e_{3,5}$ or $e_{4,6}$ should be in the Delaunay triangulation. Clearly it can not be both since they cross. But the ball with boundary through s_3, s_4, and s_6 would contain s_5, so the face $f_{3,4,6}$ cannot be in the Delaunay triangulation. On the other hand the ball with boundary through s_3, s_6, and s_5 does not contain s_4 or any other points in S, so the face $f_{3,5,6}$ is in the Delaunay triangulation.

8.1.2 Connection to Clustering

So what is the connection to clustering? Given a large set $X \subset \mathbb{R}^d$ of size n, we would like to find a set of k sites S (post office locations) so that each point $x \in X$ is near some post office. This is a proxy problem. So given a set of sites S, determining for each $x \in X$ which site is closest is exactly determined by the Voronoi diagram.

8.2 k-Means Clustering

Probably the most famous clustering formulation is k-means. The term "k-means" refers to a problem formulation, *not an algorithm*. There are many algorithms with aim of solving the k-means problem formulation, exactly or approximately. We will mainly focus on the most common: Lloyd's algorithm. Unfortunately, it is common in the literature to see "the k-means algorithm," this typically should be replaced with Lloyd's algorithm.

k-Means is in the family of *assignment-based clustering*. Each cluster is represented by a single point, to which all other points in the cluster are "assigned." The input is a data set X, and the output is a set of centers $S = \{s_1, s_2, \ldots, s_k\}$. This implicitly defines a set of clusters where $\phi_S(x) = \arg\min_{s \in S} \|x - s\|$ (the same as in the post office problem). Then the *k-means clustering problem* is to find the set S of k clusters to minimize

$$\mathsf{cost}(X, S) = \sum_{x \in X} \|\phi_S(x) - x\|^2.$$

So we want every point assigned to the closest site, and want to minimize the sum of the squared distance of all such assignments.

8.3 Lloyd's Algorithm

When people think of the k-means problem, they usually think of the following algorithm. It is usually attributed to Stuart P. Lloyd from a document in 1957, although it was not published until 1982.[1]

Algorithm 8.3.1 Lloyd's Algorithm for k-Means Clustering

Choose k points $S \subset X$ *arbitrarily?*
repeat
  for all $x \in X$: assign x to X_i for $s_i = \phi_S(x)$ *the closest site $s \in S$ to x*
  for all $s_i \in S$: set $s_i = \frac{1}{|X_i|} \sum_{x \in X_i} x$ *the average of $X_i = \{x \in X \mid \phi_S(x) = s_i\}$*
until (the set S is unchanged)

Convergence. If the main loop has R rounds, then this takes roughly Rnk steps (and can be made closer to $Rn \log k$ with faster nearest neighbor search in some cases). But how large is R; that is, how many iterations do we need?

First, we can argue that the number of steps is finite. This is true since the cost function $\mathsf{cost}(X, S)$ will always decrease. To see this, writing it as a sum over S.

$$\mathsf{cost}(X, S) = \sum_{x \in X} \|\phi_S(x) - x\|^2 = \sum_{s_i \in S} \sum_{x \in X_i} \|s_i - x\|^2.$$

[1] Apparently, the IBM 650 computer Lloyd was using in 1957 did not have enough computational power to run the (very simple) experiments he had planned. This was replaced by the IBM 701, but it did not have quite the same "quantization" functionality as the IBM 650, and the work was forgotten. Lloyd was also worried about some issues regarding the k-means problem not having a unique minimum.

Then in each step of the **repeat-until** loop, thus must decrease. The first step holds since it moves each $x \in X$ to a subset X_i with the corresponding center s_i closer to (or the same distance to) x than before. So for each x the term $\|x - s_i\|$ is reduced (or the same). The second step holds since for each inner sum $\sum_{x \in X_i} \|s_i - x\|^2$, the single point s_i which minimizes this cost is precisely the average of X_i. So reassigning s_i as described also decreases the cost (or keeps it the same).

Importantly, if the cost decreases each step, then it cannot have the same set of centers S on two different steps, since that would imply the assignment sets $\{X_i\}$ would also be the same. Thus, in order for this to happen, the cost would need to decrease after the first occurrence, and then increase to obtain the second occurrence, which is not possible.

Since, there are finite ways each set of points can be assigned to different clusters, then, the algorithm terminates in a finite number of steps.

... but in practice usually we may run for $R = 10$, or maybe $R = 20$ steps. Or check if the change in cost function is below some sufficiently small threshold.

On clusterability: When data is easily clusterable, most clustering algorithms work quickly and well. When is not easily clusterable, then no algorithm will find good clusters.

Sometimes there is a good k-means clustering, but it is not found by Lloyd's algorithm. Then we can choose new centers again (with randomness), and try again.

Initialization. The initial paper by Lloyd advocates to choose the initial partition of X into disjoint subsets $X_1, X_2, \ldots, X_k$ *arbitrarily*. However, some choices will not be very good. For instance, if we randomly place each $x \in X$ into some x_i, then (by the central limit theorem) we expect all $s_i = \frac{1}{|X_i|} \sum_{x \in X_i} x$ to all be close to the mean of the full data set $\frac{1}{|X|} \sum_{x \in X} x$.

A bit safer way to initialize the data is to choose a set $S \subset X$ at random. Since each s_i is chosen separately (not as an average of data points), there is no centering phenomenon. However, even with this initialization, we may run Lloyd's algorithm to completion, and find a sub-optimal solution (*a local minimum!*). Thus, it is usually safer to randomly re-initialize the algorithm several times (say 3-5 times) and rerun Lloyd's algorithm for each. the probability all random restarts results in a local minimum is more rare.

A more principled way to choose an initial set S is to use an algorithm like Gonzalez (for k steps, iteratively chose the point $x \in X$ furthest from all sites chosen so far), or k-means++ (for k steps, iteratively choose a point at random proportional to the squared distance to its nearest already chosen site). These are beyond the scope of this class, but offer much stronger error guarantees of various forms. But, for k-means++, it is still suggested to try random restarts.

Number of clusters So what is the right value of k? Like with PCA, there is no perfect answer towards choosing how many dimensions the subspace should be. When k is not given to you, typically, you would run with many different values of k. Then create a plot of $\mathsf{cost}(S, X)$ as a function of k. This cost will always decrease with larger k; but of course $k = n$ is of no use. At some point, the cost will not decrease much between values (this implies that probably two centers are used in the same grouping of data, so the squared distance to either is similar). Then this is a good place to choose k.

8.3.1 Extensions to the k-Means Problem and Lloyd's Algorithm

Like many algorithms in this class, Lloyd's depends on the use of a SSE cost function. In particular, it works because for $X \in \mathbb{R}^d$, then

$$\frac{1}{|X|} \sum_{x \in X} x = \mathsf{average}(X) = \arg\min_{s \in \mathbb{R}^d} \sum_{x \in X} \|s - x\|^2.$$

There are not similar properties for other costs functions, or when X is not in $\mathbb{R}^d$. For instance, one may want to solve the k-medians problem where one just minimizes the sum of (non-squared) distances. In particular, this has no closed form solution for $X \in \mathbb{R}^d$ for $d > 1$.

An alternative to the averaging step is to choose

$$s_i = \arg\min_{s \in X_i} \mathtt{d}(x, s)$$

where $\mathtt{d}(x, s)$ is an arbitrary measure (like non-squared distance) between x and s. That is, we choose an s from the set X_i. This is particularly useful when X is in a non-Euclidean metric space where averaging may not be well-defined. For the specific case where $\mathtt{d}(x, s) = \|x - s\|$ (for the k-median problem), then this variant of the algorithm is called *k-mediods*.

Soft clustering. Sometimes it is not desirable to assign each point to exactly one cluster. Instead, we may split a point between one or more clusters, assigning a fractional value to each. This is known as *soft clustering* whereas the original formulation is known as *hard clustering*.

There are many ways to achieve a soft clustering. For instance, consider the following Voronoi diagram-based approach called based on natural neighbor interpolation (NNI). Let $V(S)$ be the Voronoi diagram of the sites S (which decomposes $\mathbb{R}^d$). Then construct $V(S \cup x)$ for a particular data point x; the Voronoi diagram of the sites S with the addition of one point x. For the region R_x defined by the point x in $V(S \cup x)$, overlay it on the original Voronoi diagram $V(S)$. This region R_x will overlap with regions R_i in the original Voronoi diagram; compute the volume v_i for the overlap with each such region. Then the fractional weight for x into each site s_i is defined $w_i(x) = v_i / \sum_{i=1}^n v_i$.

We can plug any such step into Lloyd's algorithm, and then recalculate s_i as the weighted average of all points partially assigned to the ith cluster.

8.4 Mixture of Gaussians

The k-means formulation tends to define clusters of roughly equal size. The squared cost discourages points far from any center. It also, does not adapt much to the density of individual centers.

An extension is to fit each cluster X_i with a Gaussian distribution $\mathcal{G}(\mu_i, \Sigma_i)$, defined by a mean μ_i and a covariance matrix Σ_i. Recall that the pdf of a d-dimensional Gaussian distribution is defined

$$f_{\mu,\Sigma}(x) = \frac{1}{(2\pi)^{d/2}} \frac{1}{\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

where $|\Sigma|$ is the determinant of Σ. For instance, for $d = 2$, and the standard deviation in the x-direction of X is σ_x, and in the y-direction is σ_y, and their correlation is ρ, then

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix}.$$

Now the goal is, given a parameter k, find a set of k pdfs $F = \{f_1, f_2, \ldots, f_k\}$ where $f_i = f_{\mu_i,\Sigma_i}$ to maximize

$$\prod_{x \in X} \max_{f_i \in F} f_i(x),$$

or equivalently to minimize

$$\sum_{x \in X} \min_{f_i \in F} -\log(f_i(x)).$$

For the special case where when we restrict that $\Sigma_i = I$ (the identity matrix) for each mixture, then one can check that the second formulation (the log-likelihood version) is equivalent to the k-means problem.

This hints that we can adapt Lloyds algorithm towards this problem as well. To replace the first step of the inner loop, we assign each $x \in X$ to the Gaussian which maximizes $f_i(x)$:

$$\text{for all } x \in X\text{: assign } x \text{ to } X_i \text{ so } i = \arg\max_{i \in 1 \dots k} f_i(x).$$

But for the second step, we need to replace a simple average with an estimation of the best fitting Gaussian to a data set X_i. This is also simple. First, calculate the mean as $\mu_i = \frac{1}{|X_i|} \sum_{x \in X_i} x$. Then calculate the covariance matrix Σ_i of X_i as the sum of outer products

$$\Sigma_i = \sum_{x \in X_i} (x - \mu_i)(x - \mu_i)^T.$$

This can be interpreted as calling PCA. Calculating μ_i, and subtracting from each $x \in X_i$ is the centering step. Letting $\bar{X}_i = \{x \in \mu_i \mid x \in X_i\}$, then $\Sigma_i = VS^2V^T$ where $[U, S, V] = \mathsf{svd}(\bar{X}_i)$.

8.4.1 Expectation-Maximization

The standard way to fit a mixture of Gaussians actually uses a soft-clustering.

Each point $x \in X$ is given a weight $w_i = f_i(x) / \sum_i f_i(x)$ for its assignment to each cluster. Then the mean and covariance matrix is estimated using weight averages.

Algorithm 8.4.1 EM Algorithm for Mixture of Gaussians

Choose k points $S \subset X$ — *arbitrarily?*
for all $x \in X$: set $w_i(x)$ for $s_i = \phi_S(x)$, and $w_i(x) = 0$ otherwise
repeat
  for $i \in [1 \dots k]$ **do**
    Calculate $W_i = \sum_{x \in X} w_i(x)$ — *the total weight for cluster i*
    Set $\mu_i = \frac{1}{W_i} \sum_{x \in X} w_i(x) x$ — *the weighted average*
    Set $\Sigma_i = \frac{1}{W_i} \sum_{x \in X} w_i(x - \mu_i)(x - \mu_i)^T$ — *the weighted covariance*
  for $x \in X$ **do**
    for all $i \in [1 \dots k]$: set $w_i(x) = f_i(x) / \sum_i f_i(x)$ — *partial assignments using $f_i = f_{\mu_i, \Sigma_i}$*
until ($\sum_{x \in X} \sum_{i=1}^{k} - \log(w_i(x) \cdot f_i(x))$ has small change)

This procedure is the classic example of a framework called *expectation-maximization*. This is an alternate optimization procedure, which alternates between maximizing the probability of some model (the partial assignment step) and calculating the most likely model using expectation (the average, covariance estimating step).

But this is a much more general framework. It is particularly useful in situations (like this one) where the true optimization criteria is messy and complex, often non-convex; but it can be broken into two or more steps where each step can be solved with a (near) closed form. Or if there is no closed form, but each part is individually convex, the gradient descent can be invoked.

8.5 Mean Shift Clustering

Now for something completely different. Clustering is a very very broad field with no settled upon approach. To demonstrate this, we will quickly review an algorithm called *mean shift clustering*. This algorithm shifts

each data point individually to its weighted center of mass. It terminates when all points converge to isolated sets.

First begin with a bivariate kernel function $K : X \times X \to \mathbb{R}$ such as the (unnormalized) Gaussian kernel

$$K(x, p) = \exp(-\|x - p\|^2/\sigma^2)$$

for some given bandwidth parameter σ. The weighted center of mass around each point $p \in X$ is then defined as

$$\mu(p) = \frac{\sum_{x \in X} K(x, p)x}{\sum_{x \in X} K(x, p)}.$$

The algorithm just shifts each point to its center of mass: $p \leftarrow \mu(p)$.

Algorithm 8.5.1 Mean Shift

repeat

for all $p \in X$: calculate $\mu(p) = \frac{\sum_{x \in X} K(x,p)x}{\sum_{x \in X} K(x,p)}$.

for all $p \in X$: set $p \leftarrow \mu(p)$.

until (the average change $\|p - \mu(p)\|$ is small)

This algorithm does not require a parameter k. However, it has other parameters, most notably the choice of kernel K and its bandwidth σ. With the Gaussian kernel (since it has infinite support, $K(x, p) > 0$ for all x, p), it will only stop when all x are at the same point. Thus the termination condition is also important. Alternatively, a different kernel with bounded support may terminate automatically (without a specific condition); for this reason (and for speed) truncated Gaussians are often used.

This algorithm not only clusters the data, but also is a key technique for *de-noising* data. This is a process that not just removes noise (as often thought of as outliers), but attempts to return point to where they should have been before being perturbed by noise – similar to mapping a point to its cluster center.