Mean-shifting Clustering

Juan David Rodríguez Cuervo

March 2018

The **Mean-shifting algorithm** is a method to classify data points in an n-dimensional dataset in centroid-based clusters. Different to the k-means algorithm, the mean-shifting does not presume the number of clusters to make. That is, in the algorithm is internally decided how many clusters are there to be made. Another big difference (which has a greater implication in efficiency), is that to move each centroid in every step, it does not use the mean-value of all the points in each cluster. Instead, it defines a radius region around the centroid, and updates the centroid's position based on the mean values of the points in the radius region for each feature.

As a concept, it starts with a matrix of centroids. Each centroid moves to the closest region with highest concentration of data points. If a centroid after update has very few points falling in the region, it is "forgotten". This is because if a centroid is in a low-density area, its movement will be despicable, and instead of moving to a high density area, it will remain on a noise area, where it will create a cluster that does not describe the dataset.

In order to understand the algorithm, it is necessary to introduce the distance function, defined as usual in the euclidean space:

$$d(p,q) = \sqrt{\sum_{n=1}^{f}(p_n - q_n)^2} \tag{1}$$

This distance function is going to serve on another criterion to neglect a centroid. Additional to the low-density space in which a centroid might have entered on any update, another way a centroid is considered unnecessary and thus deleted, is when, on position update, it is *very close* to another centroid.

Let C_n be the set of centroids at learning iteration n, and $k_n = |C_n|$ the number of clusters in C_n. It can be seen that k_n might be different to k_m for $n \neq m$, as on each learning iteration, some centroids might be deleted i.e. when they are in a very low density area, or their position is close to another centroid's position.

To neglect a centroid means that it will no longer be part of the list of "active" centroids; it will be ignored, or eliminated. Thus, its position will no longer be updated and so. In other words, the criterion states that a centroid c_i will stop existing if:

$$\exists c_j \neq c_i : d(c_i', c_j') < r \tag{2}$$

Here, $r \in \mathbb{R}$ is a constant that defines the minimum distance accepted for the centroids to be together; $c_i, c_j \in C_n$, and $c_j' \in C_{n+1}$. In equation 2, c_i' is used to denote the centroid c_i right after its position has been updated. As it can be seen, c_i' may belong or not to C_{n+1}, depending on the criteria, but every c_j to which it is compared has the restriction that $c_j' \in C_{n+1}$.

This algorithm, ideally, runs until it reaches a stable state in the whole system. Nonetheless, it is also added a maximum number of iterations or steps to take, to avoid very long execution times. This means that in order to stop the execution of the program, it either has to reach a maximum step number, or reach a convergence state in every centroid of the system in a given learning step. It is said with respect to a given centroid that it has reached a convergence state if the position from one step to another (that is, during a certain step) is the same. In other words, the distance of the original centroid, and the updated centroid is zero, or:

$$d(c_i, c_i') = 0 \tag{3}$$

When this is true for all centroids in a given learning iteration, the program must stop. This is because the new position is given as the mean point in the region, and if the region of a centroid remains unchanged, then the data points in it will also be the same, and so the position will not change. Therefore, from that convergence point, the system will remain unchanged for further update steps.

It still remains missing the way each centroid's position is updated on every learning iteration. After a position is defined, the update process is done by selecting the points of the dataset which fall within a region near the centroid. This region can be implemented as a circular region (radius $= \epsilon$), but in the present code it is done as a square region s (side length $= 2\epsilon$). Having s dataset D with F features, then:

$$s(c_i) = \{p \in D : c_i(f) - \epsilon \leq p(f) \leq c_i(f) + \epsilon, \quad \forall f \in F\} \tag{4}$$

After this region is defined, then the updated position of the centroid is given by:

$$c_i'(f) = mean(s(f)), \quad \forall f \in F \tag{5}$$

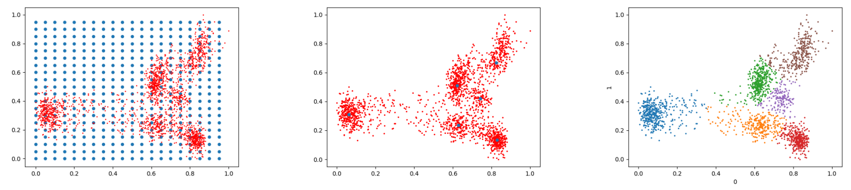
This also means that the first criterion mentioned to decide whether a centroid remains or is deleted, is given in the form:

$$c_i' \in C_n \leftrightarrow |s(c_i)| > t \tag{6}$$

with t being the threshold value chosen as the minimum number of data points accepted in a region.

1 Results

The algorithm was coded in Python and run on some test datasets. It was programmed to show the first setting, that is, the dataset points with the initial matrix of centroids, then show the same points but with the final centroids' positions, and finally show the dataset points as classified with respect to the clusters chosen by the system. All this process can be seen in Figure 1.



(a) Initial centroid setting (b) Centroid positioning (c) Data points classified

Figure 1: Visual Process of learning

It is important to highlight the fact that the system automatically chooses the number of clusters to be left, depending on the dataset. As with efficiency, running with datasets with 2k entries, 2 features (i.e. 2 dimensional datasets, for visualization), and an initial configuration of 400 centroids, it barely takes 2 seconds to process.

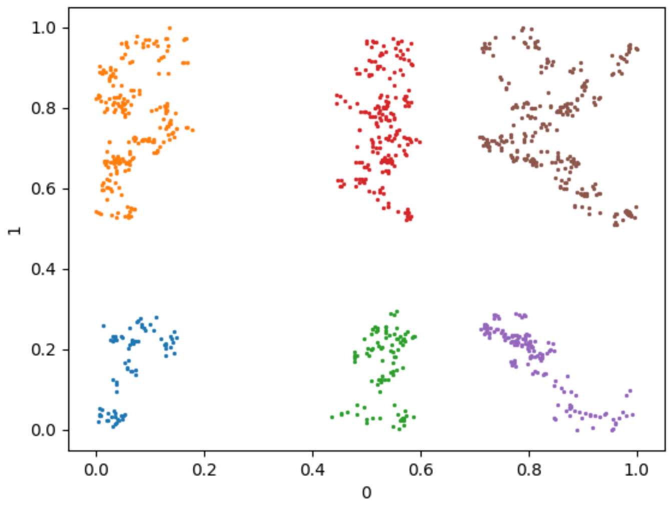


Figure 2: Classification in 6 clusters