K-means improvement and study of image sequencing

Elisa Drouot[1]

Abstract

We know that the k-means algorithm has drawbacks: it is sensitive to the shape of clusters; if clusters have complex geometric shapes, k-means may yield poor results. Additionally, it is sensitive to the choice of initial parameters, which can alter the outcome. It is also sensitive to outliers. That's why I chose to study an improved version called ddk-means, as presented in the article 'An Improved K-means Clustering Algorithm' by Meriem K. and Cérine-Maroua B. This article introduces the method for 2D data, whereas the work presented here is specifically tailored for 3D data. I will present how to use the basic kmeans algorithm to sequence images and comparing the efficiency of the new algorithm to the efficiency of the basic one in image sequencing.

Keywords : **Clustering, DDKmeans, Image sequencing, Kmean, Python**

1. Image sequencing with kmeans algorithm

The principle of image sequencing using k-means involves selecting a number of clusters (this number determines the number of different regions desired in our sequencing) and then grouping pixels by clusters. A pixel is represented by a list with 3 elements, each representing a color: red, green, and blue. Therefore, each pixel will be in the form: [R, G, B].

Figure 1: Images used in this work

So has we can see in annexes, I segmented the two images for values of k ranging from 2 to 11. We can see that the segmentation is very well made on those images.

The main difficulty here is to find the color of each cluster, I have made functions for this. To find the color of a cluster I calculate the median of all the pixel in the cluster. After this I create a new image and each point in the cluster will take the value of the median that has been calculated previously.

2. DDK-means

The DDK-means is an improved version of K-means that focuses on addressing two major weaknesses of the K-means algorithm, namely, the elimination of outliers and the selection of relevant initial centroids.Because if the initial centroids are poorly chosen, the algorithm may converge to a solution that is far from the global optimum

To achieve this, outliers are first identified using the isolation forest algorithm, which involves partitioning the space until a point is found to be alone. For instance, in the image below, it can be observed that the point in the top-right corner is an outlier as it ends up alone in a zone after four splits:

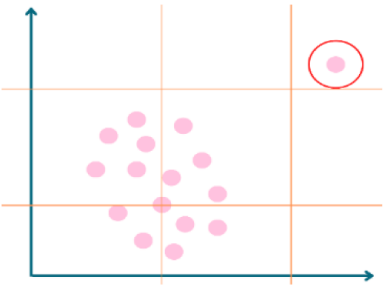

Figure 2: Isolation forest algorithm illustration

We can observe the isolation forest on a Swiss roll-type dataset. In the following image, the first graph shows the outliers in black. Once these outliers are identified, they will be replaced with the preceding value in the dataset (ideally, averaging with neighboring points would be optimal, but this is more complex and costly). The corrected points are illustrated in the second graph as green points:

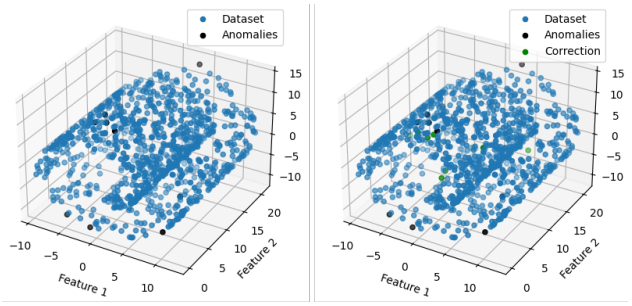

Figure 3: Isolation forest algorithm on swiss roll

Next, we will apply DDK-means to another dataset containing two outliers, represented as follows (with the two outliers in black):

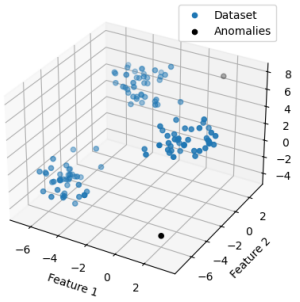

Figure 4: Isolation forest algorithm on another dataset

It is interesting to note that the removal of outliers facilitates the selection of good centroids: indeed, K-means chooses the initial centroids randomly. Consequently, the presence of anomalies in the dataset increases the probability that one of them will be selected as the initial centroid.

Now, we tackle the search for relevant initial centroids, which is divided into three steps.

The first step involves calculating the diagonal. To find the value of this diagonal, we compute the distance between two opposite points in the dataset. Here, we calculate the distance between (xmax, ymin, zmin) and (xmin, ymax, zmax). For our dataset, we obtain the following diagonal:

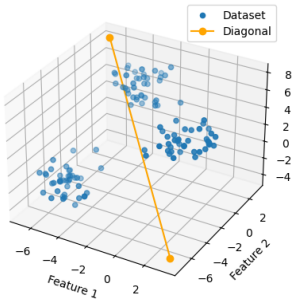

Figure 5: Calculation of the diagonal on the used dataset

The second step involves calculating alpha, which represents the value of the diagonal divided by the number of clusters in the database. This value will be used later as a neighborhood radius. The drawback here is that it requires knowing the number of clusters; however, in image sequencing, there is not a precise number of clusters. The number of clusters depends on how one wants to sequence the image, so the number of clusters must be an input parameter. For example, for the second image, if we plot the inertia as a function of k, we can see that there is no specific elbow, indicating no optimal number of clusters:

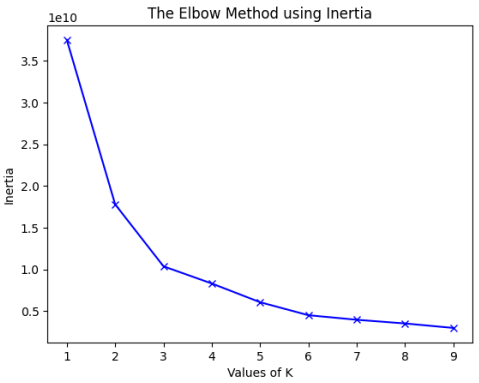

Figure 6: Inertias as a function of k of the second image

Alpha is calculated as follows:

$$\alpha = \frac{\text{diagonal}}{k}$$

Where:

α: neighborhood radius

k: the number of clusters to be formed (predefined)

diagonal: calculated in the previous step.

The third step involves calculating the density of each point in the dataset. The objective here is to ultimately select centroids with significant density, meaning points that are not outliers. The density of each point represents the number of neighbors whose distance to that point is less than alpha. For example, in the following diagram, the green lines connect the point to those considered its neighbors, while the red lines connect the points that will not be considered its neighbors.

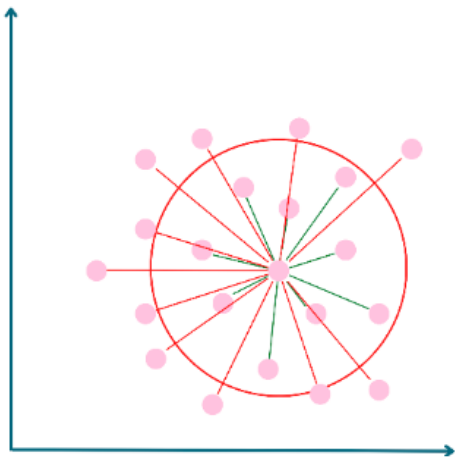

Figure 7: Found centroids of the dataset

After this I retrieve the $x * k$ index of points that have the biggest density. Then I make the combination of all set of three point possible and calculate the distance between those three points. The set that have the biggest distance is the three centroids.

For example on my dataset I obtain the three oranges points has centroids :

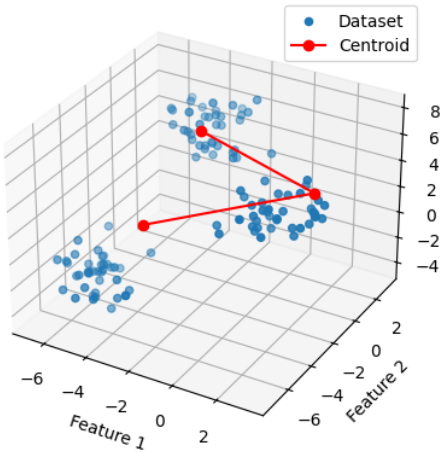

Figure 8: Found centroids of the dataset

All the steeps behind can be resumes has the following algorithm :

Algorithm:
Input: datset, k
Output: DDK-means model

1 1.Isolation Forest to detect anomalies `/* Returns AnomalyList */`
2 **for** *each pixel*[*i*] *in AnomalyList* **do**
  `/* with i ranging from 0 to the size of the image */`
3 *pixel*[*i*] = *pixel*[*i* − 1]
4 **end**
5 2.Calculate diagonal
  = distance((xmax, ymin, zmin), (xmin, ymax, zmax))
  /* $\sqrt{(xmax - xmin)^2 + (ymin - ymax)^2 + (zmin - zmax)^2}$ */
6 3.Calculate density $\alpha = \frac{\text{diagonal}}{k}$
7 $D = []$
8 *index* = []
9 **for** *each pixel*[*i*] *in the image* **do**
10 *neighbors* = 0 **for** *each pixel*[*j*] *in the image* **do**
11 **if** *distance*(*pixel*[*i*], *pixel*[*j*]) ≤ α **then**
12 *neighbors* + +
13 **end**
14 **end**
15 *D*.append(*neighbors*)
16 *D*.append(*i*)
17 **end**
18 **if** *D*[*i*] *is among the top* 5*k maximum values* **then**
19 Keep its index to create a new index list of size 5*k*
20 **end**
21 4.Generate combinations of all combinations with *k* elements, resulting in possible sets *max* = 0
  centroids = [] **for** *all possible sets* **do**
22 **if** *the distance between the k points of the set* > *max* **then**
23 *max* = distance between the *k* points of the set
  centroids = points of the set
24 **end**
25 **end**
26 5.Apply classical k-means using the selected centroids

3. Conclusion

In conclusion, I would say that mage segmentation using k-means is quite interesting but remains challenging to use. K-means is, in fact, an algorithm with numerous flaws, making it compelling to explore improvement avenues such as DDK-means, which allows us to obtain relevant centroids. In conclusion, I would say that classification is a unique discipline in Machine Learning that continues to evolve, and it is this constant improvement that makes it so intriguing.

Annexes

Image 2 segmentée avec K = 2

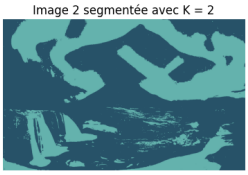

Image 2 segmentée avec K = 3

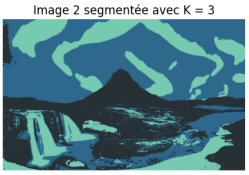

Image 2 segmentée avec K = 4

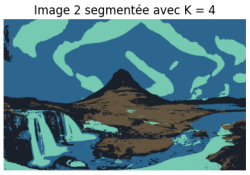

Image 2 segmentée avec K = 5

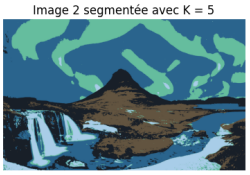

Image 2 segmentée avec K = 6

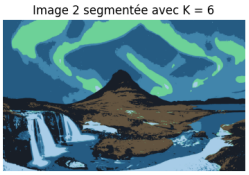

Image 2 segmentée avec K = 7

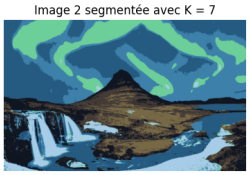

Image 2 segmentée avec K = 8

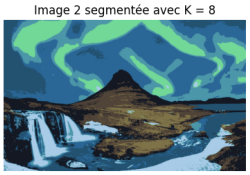

Image 2 segmentée avec K = 9

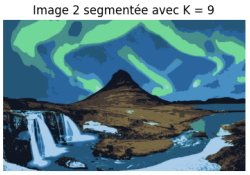

Image 1 segmentée avec K = 2

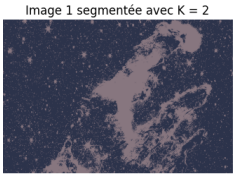

Image 1 segmentée avec K = 3

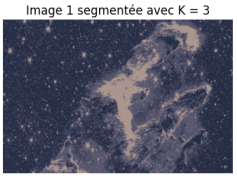

Image 1 segmentée avec K = 4

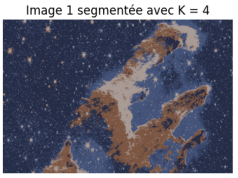

Image 1 segmentée avec K = 5

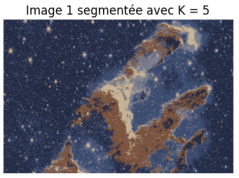

Image 1 segmentée avec K = 6

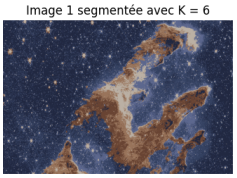

Image 1 segmentée avec K = 7

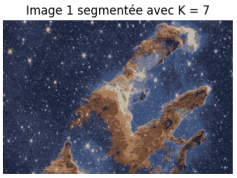

Image 1 segmentée avec K = 8

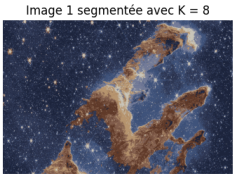

Image 1 segmentée avec K = 9