K-Means Clustering

A clustering algorithm is an algorithm designed with the purpose of grouping, into an arbitrary number of classes, a set of points. Such classes are created by to grouping together those points that have common features. Thus, the goal of this category of algorithms is to build knowledge about the data provided as input that makes it possible to find hidden relationships among them which can be used to classify (into one of the classes created in the training stages) a point whose class is not known a priori.

Standard (or Lloyd) Algorithm

In this section, the standard algorithm for implementing k-means clustering is introduced.
First, it is important to define the initial condition (i.e. the input data) on which the algorithm will be run. Considering the case in two dimensions, the input is a collection of points that are randomly placed in 2D space (Figure 1).

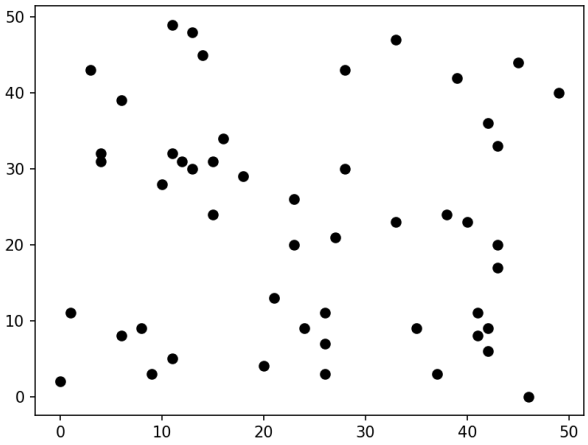

Figure 1: *Initial distribution of random points*

The first step is to generate an arbitrary number of centroids, i.e. points in space representing the classes to which points will be assigned according to a criterion of maximum closeness.

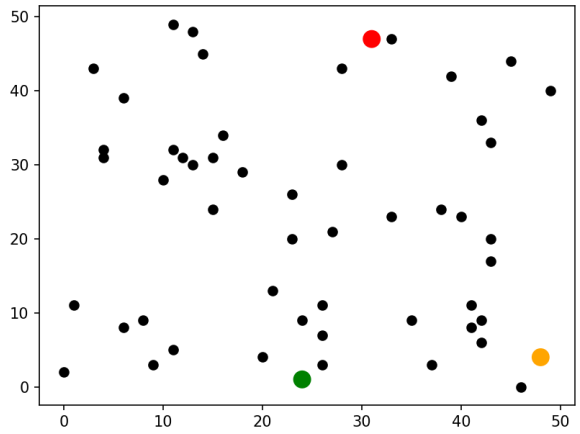

Figure 2: *Random initial position of centroids*

Figure 2 shows three centroids representing 3 classes (red, green, orange). The generation of centroids can be done in 2 ways: by choosing some of the points provided as input, or by randomly generating them in space.
After that the centroids are generated, each point is assigned a class according to the principle of maximum closeness.

Given a point P = (x, y) in space its membership class is the one represented by the centroid whose distance from the point is less than that of all other centroids. The choice of how to measure the distance between two points in space affects the final result of the classification. Therefore it is necessary to specify which measure is being used, in this case the Euclidean distance. Given two points A = (a_x, a_y) and B = (b_x, b_y) the Euclidean distance is calculated as

$$d(A, B) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} \tag{1}$$

Figure 3 shows the points after they have been assigned to the closest centroid.

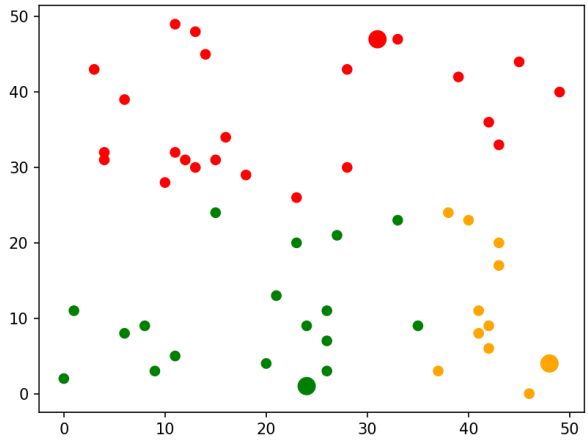

Figure 3: *Points after the assignation*

The next step is to reposition the centroids by calculating the average of all points belonging to its class. If C_i is the i-th centroid, its new position ($\tilde{C}_i$) can be calculated as

$$\tilde{C}_i = \frac{1}{N_i} \cdot \sum_{p \in C_i} p \tag{2}$$

Where N_i is the number of points belonging to the i-th class. Figure 4 shows the centroids after being repositioned.

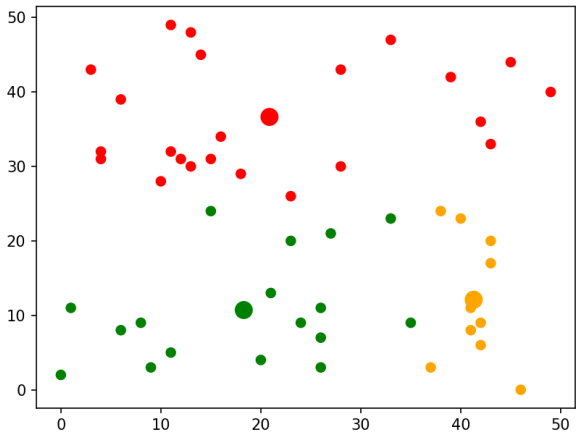

Figure 4: *Centroids after being recalculated*

Now the point assignation is repeated and then the centroid recalculation and so on until a convergence criterion is reached, which for example could be when the centroids do not shift more than a certain threshold, or simply after a certain number of iterations. This latter method could be usefull when comparing two different algorithm (like lloyd vs hamerly algorithm which is described later)

because the two could converge after a different number of iterations therefore by using a fixed number of repetitions it is easy to compare the result after the same number of cycles. Because of this, for this report, this second approach has been used.

Number of iterations

The iterations required during a single cycle are

$$N \cdot K \cdot d$$

where N is the number of points, K is the number of centroids and d is the dimensionality of the data. This is because for each point it is necessary to calculate the distance to each centroid to know which is the least, and to calculate the distance it is necessary to sum k terms (one for each dimension).
So if m is the number of cycles, the iterations needed are in total

$$m(N \cdot K \cdot d) \tag{3}$$

Algorithm 1 k-means pseudo-code

```
Let "points" be the set of all points
Let "centroids" be the set of all centroids
for i < iterations do
                                        ▷ Point assignation
    for p in points do
        mindist = min_j d(p, C_j)
        old_centroid = p.centroid_index
        p.centroid_index = j

        for l < d do
            coord = p.get_coord(l)
            average_per_class[j][l] += coord
            if old_centroid ≠ -1 then
                average_per_class[old_centroid][l] -= coord
            end if
        end for

        average_per_class[j] += 1
        if p.centroid_index ≠ -1 then
            average_per_class[old_centroid] -= 1
        end if
    end for
                                        ▷ Centroid update
    for c in centroids do
        nPoints = points_per_class[i]
        for j < d do
            new_coord = average_per_class[i][j] / nPoints
            c.set_coord(j, new_coord)
        end for
    end for
end for
```

Tracking points belonging to a class

During the point assignation step, an important operation is made: tracking which points belong to the classes. In particular we are referring to the following part of the code

Algorithm 2 k-means pseudo-code

```
for l < d do
    coord = p.get_coord(l)
    average_per_class[j][l] += coord
    if old_centroid ≠ -1 then
        average_per_class[old_centroid][l] -= coord
    end if
end for

average_per_class[j] += 1
if p.centroid_index ≠ -1 then
    average_per_class[old_centroid] -= 1
end if
```

Since this is an important process and might not be instantly clear, in this section we will go a little bit deeper and explain how the point tracking is done.
First of all the average_per_class matrix, which is a Kxd matrix that contains, for each row, the sum of all coordinates of the points belonging to that class.
Then the points_per_class array, which is an array with a length of K and, for each class, it contains the number of points which are contained in that class.

Thus, when a point is assigned to the j-th class, its coordinates are added to the average_per_class[j] row of the matrix while points_per_class[j] is incremented by one. It is also important to remove the point from the previous class if it was previously assigned to one. The way to verify it is through the condition $p.centroid\_index! = -1$. If the condition is true it means that the points had a previous class. After the verification of the condition, to remove the point from its previous class, its coordinate need to be subtracted from the average_per_class[p.centroid_index] row and points_per_class[p.centroid_index] need to be decremented by one.

Thanks to this process which is done during the points assignation, the following step (i.e. centroid update) is much easier to do. In fact, it is enough to go through the matrix lines and divide it by the total number of points in that class which is contained in the points_per_class array.