CLUSTERING ALGORITHM FOR CUSTOMERS SEGMENTATION

GIACOMO BARILARO

BENEDETTO CAVICCHI

1. INTRODUCTION

In any sector, customers are the most important part of the business life cycle. It is essential for any analyst or data scientist professional to know better about them. Customer segments enable you to understand the patterns that differentiate your customers. Analyzing customer segments you to customize your offer. We realized this purpose with clustering algorithms.

A *clustering algorithm* takes as input a set of data points associated with their attributes and produces as output a partition of the input data in groups called *clusters*. In the end, each observation is labeled with the identifier of the cluster to which it was assigned. Partitioning is produced by analyzing the input data, comparing the sample properties and finding the most relevant similarities among them. In this project we will consider only clustering algorithms that generate non-overlapping clusters, i.e., each point is assigned to one single cluster. The reason why we chose Clustering method is because, differently to other kind of supervised classification algorithms, it allows to treat domains which are not characterized by group labels that are known in advanced. The aim of this paper is to find the best algorithm to perform this kind of analysis, with respect to fitting the data and computational costs.

The data set that we will consider is called *Mall Customer Segmentation Data*. This data set is created only for the learning purpose of the customer segmentation concepts. The situation reproduced is the following: you own a supermarket mall and through membership cards, you retrieve relevant data about your customers like Customer ID, age, gender, annual income and spending score. Spending Score is an index that is assigned to the customer based on their purchasing behaviour. For simplicity we'll implement the analyses on the most two significative dimensions: Annual Income and Spending Score.

First of all we will describe the clustering method that we used to compare the different algorithms. For each algorithms we will compute the time complexity, the silhouette score and we will plot the results. The interrelation of these three different dimensions will allow us to decide which algorithm is the best to segment customer of our dataset.

2. CLUSTERING METHOD

CLUSTERING METRICS

The clustering metric is the method to measure the distance between a pair of samples. It is a fundamental function because it establishes the criterion with which the algorithm to compares the data points and the chosen metric highly influences the clusters shape. The cluster metric we chose for our analyses is the Euclidian distance beween points a, b, defined as follow:

$$\|a-b\|_2 = \sqrt{\sum_i (a_i - b_i)^2}$$

Where a_i and b_i are the values of samples a and b for attribute i. It is reasonable using Euclidian distance because of the domain of the two points is R^2.

CLUSTERING ALGORITHMS

The algorithms that we perform belong to the most important families of clustering algorithms: partitional, hierarchical ad density-based. In particular we use the standard version of each of these algorithm.

K MEANS

K means algorithm is partitional since it uses as parameter the desired number of clusters *k,* i.e. doesn't produce the number of clusters as output. Each cluster is identified by a *centroid*. An observation is assigned to the cluster whose centroid is closer. K means initially sets *k* centroids with random positions. At each iteration of the algorithm data points are associated with the closest centroid. Then centroids are updated towards the center of gravity of their cluster. It is computed as the multidimensional mean of all the data points of the cluster:

$$centroid(c) = \frac{1}{|c|}\sum_{x_i \in c} x_i$$

where c is the considered cluster, $|c|$ is the number of points in c and x_i are the points associated with cluster c.

DBSCAN

This algorithm groups together points based on their density in the space. The number of generated clusters is not known a-priori and depends on the data distribution. DBSCAN takes as input two parameters to model the concept of neighborhood and density between data samples. The first one (*epsilon*) specifies the radius of the neighborhood of a point. The second one (*minpoints*) specifies the minimum number of samples in the neighborhood of a point. Samples with less than minpoints neighbors are marked as noise (outliers), which are located in low-density regions (whose nearest neighbors are too far away).

HIERARCHICAL

This algorithm generates clusters which are organized in a hierarchy. The output hierarchy is encoded with a structure called *dendrogram*. This structure specifies the aggregations of the clusters at different levels. The algorithm does not take parameters as input.
However, a user can specify *k* as the desired number of clusters in order to extract the grouping of the data samples at a particular level of hierarchy. Hierarchical algorithms works by repeatedly identifying and merging the two clusters which are the most close at a specific hierarchy level. Initially each point is considered as a specific cluster, while at the end of the hierarchy the algorithm ends with a single cluster.

CLUSTERING EVALUATION

After labeling the data points with one of the proposed clustering methods, the characteristics of each cluster can be described in order to ease the analysis of the results. In the following paragraphs we briefly describe the technique we adopted for cluster description.

SILHOUETTE

Since clustering is an unsupervised technique, measuring the quality of the results is often a difficult task. *Silhouette* is a metric defined to measure the characteristics of each cluster without having prior knowledge on the expected clusters. This measure takes values between -1 and 1. Values closer to 1 imply a good quality of the clusters. Silhouette is computed by inspecting the cohesion of the samples inside each cluster and their separation from the samples in other clusters. *Intra cluster separation* and *cluster separation* are the two components used to compute the Silhouette value. The *intra cluster separation* of a cluster c is defined as:

$$intra(c) = \sum_{x_i \in c,} \sum_{x_j \in c, i \neq j} ||x_i - x_j||^2$$

where x_i and x_j are two points of the cluster c and $||x_i - x_j||^2$ is the squared distance between the two points. Higher values of the intra cluster separation imply a lower density of the cluster, since its points are more distant among each other. The *inter cluster separation* of a cluster c is defined as follows.

$$inter(c) = \sum_{x_i \in c,} \sum_{x_j \notin c} ||x_i - x_j||^2$$

where x_i is a point of cluster c and x_j is a point of any other cluster. Higher values of the inter cluster separation imply a higher distance between the specified cluster and the other ones. The *silhouette score* gathers these two contributions. It is defined as follow:

$$silh(c) = \frac{inter(c) - intra(c)}{\max(inter(c) - intra(c))}$$

Higher values are obtained when the clusters have a relative higher inter separation with respect to the intra separation. The intra/inter cluster separation can also be computed globally for all the clusters:

$$intra = \sum_{c \in C} intra(c)$$

$$inter = \sum_{c \in C} inter(c)$$

where C is the set of all clusters.

With these definitions it is possible to compute the global silhouette:

$$silh = \frac{inter - intra}{\max(inter, intra)}$$

The global silhouette allows understanding the global quality of the produced clusters.

PLOTTING

Plotting allows us to see if the clustering of the samples yields reasonable result by compared to the distribution of the data.

3. IMPLEMENTATION

In this section we will analyze the clustering procedures described above from the point of view of the implementation. In particular, what library we used to implement the technique, the extra code we add to build the result and our consideration about issues and problems that occurred in the development process

LIBRARIES

This section describe the most important libraries we used to implement the code in Python.

Pandas. Is a library designed for the management of data structures. In fact, it provides several tools that facilitate the reading, editing and writing of relational data. It was very useful to import the datasets in the csv format and to store it content in the data structures where can be edited and analyzed.

Numpy. Is instead a fundamental package for scientific computing with Python that allows to easily manage arrays and execute functions in multi-dimensional data.

Scikit-learn. Is the main library of the project that provides all the functions to implement the classification and the clustering algorithms.

Scipy. Is another support library, useful to manage the Hierarchical strategy.

Matplotlib. This library allows to visualize all the result in plots.

K MEANS

The k means algorithm is a partitional clustering. The problem of the partitional clustering can be formally stated as follows. Given n patterns in a d-dimensional metric space, determine a partition of the patterns into K groups, or clusters, such that the patterns in a cluster are more similar to each other that to patterns in different clusters.

The partitional strategy that is used here for implementing the k means algorithm is based on the square error criterion. The general objective is to obtain the partition which, for a fixed number of clusters, minimizes the square-error. A general k means algorithm is given below.

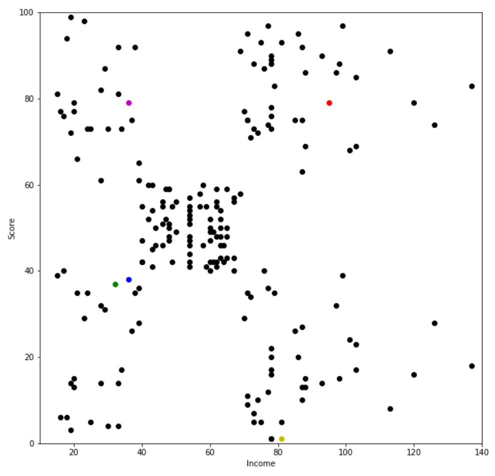
Step 1. Select an initial partition with K clusters, each of them contain an initial centroids generate randomly.

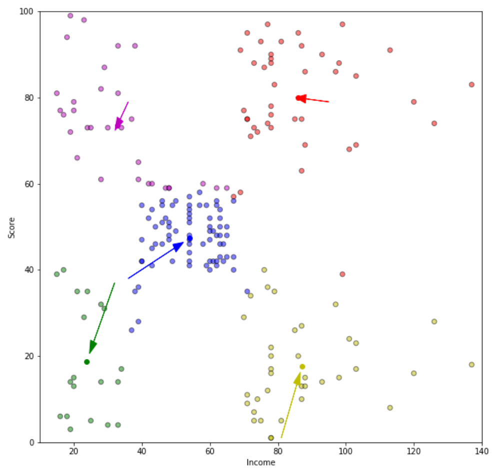
Step 2. Generate a new partition by assigning each pattern to its closed centroids.

Step 3. Compute new cluster centers as the centroids of the clusters.

Step 4. Repeats steps 2 and 3 until the square error within the clusters is minimized.

Different initial partitions can lead to different final clustering because algorithms based on the square-error can converge to a local minima. One way to overcome local minima is to run the partitional algorithm with several different initial partitions. If they all lead to the same final partition, we have some confidence that the global minimum of square-error has been achieved.

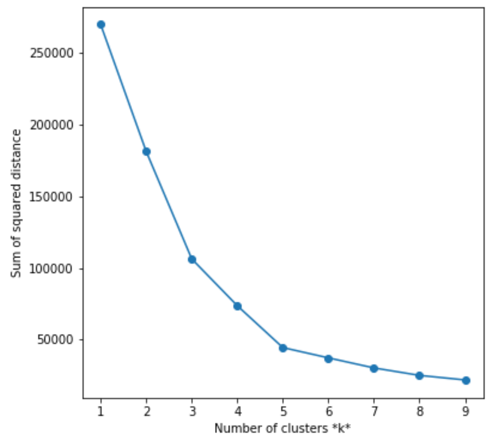




The pictures shows the iterative process

We applied the procedure to our data set below.

EXPLORATORY RESULT: Plotting the within sum of square against the number of clusters: the "elbow" gives a preliminar idea about the optimal number of clusters. In this case, the elbow is for k=5.



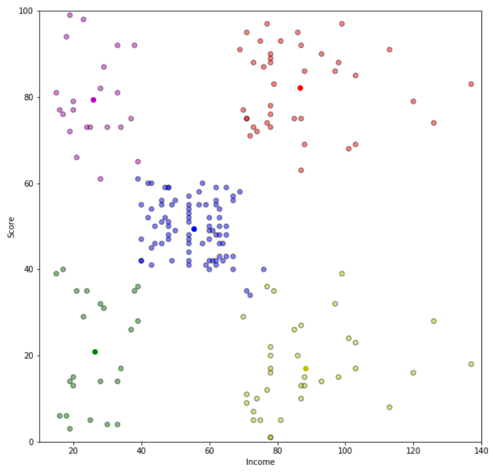
COMPUTATIONAL COSTS: The pseudo code that we used to compute the time complexity is the following.

```
Input: A set $X$ of object $(x_1, \dots, x_n)$, $k$ number of clusters
choose $k$ initial centers $C = (c_1, \dots, c_k)$
while stopping criterion has not been met
    do ⊳ assignement step:
        for i=1,..,N
        do find closest center $c_k \in C$ to
            instance $p_i$ to set $C_k$
        ⊳ update step:
        for $i = 1, \dots., k$
        do set $c_i$ to be the center of
                mass of all points in $C_i$
```

The time complexity is $O(kiN)$, where is the number of iteration to minimize the within sum of square. The number of iterations until convergence is often small. But in the worst case $i = 2^{\Omega(\sqrt{n})}$, so the worst case complexity is superpolynomial.

SILHOUETTE SCORE: It is obtained with the global silhouette on $K \in [4,6]$ (the values around the elbow in the picture) and it produces the maximum value in k=5, with silhouette score of **0.553931997444648.** The information yielded from the global silhouette confirms the previous result.

PLOT:



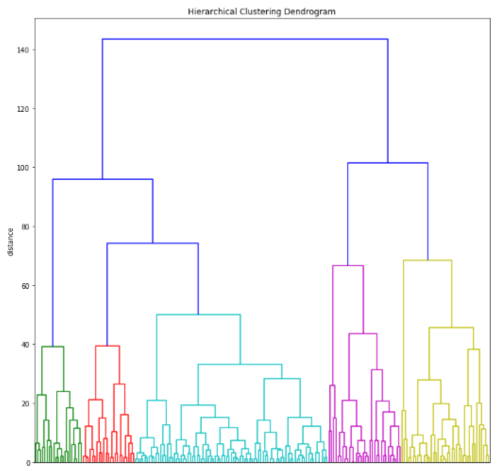
The scatter plot is divided in 5 region. The good fitting depends on the globular and well-divided distribution of the data.

The regions correspond to 5 different kind of customers: the blue clusters represents the customers with medium Income and Spending Score; the yellow cluster represents the customers with an high Income and a low Spending Score; the green cluster represents customers with low Income and low Spending Score; the violet cluster represents customers with a low Income and high Spending Score; the red cluster represents customers with high Income and high Spending Score. Despite the aim of this work it is not to do a cluster analysis, some observations about our fitting is needed. First of all, there are three groups with positive correlation between Income and Spending Score, and two clusters with negative correlation between Income and Spending Score, i.e. there are two different behaviour trends of the customers depending on their Income. Morevover, except the blue cluster, the other clusters seems have a lot of variability within the clusters. It is fundamental understanding the nature of the groups of our customers and the trends of their behaviour in order to elaborate a strategy market's for our mall.

HIERARCHICAL

To implement the Hierarchical clustering we adopted the **Agglomerative** strategy. It is a ”bottom up” approach. It means that at the beginning each sample belongs to a different cluster. Later, in each execution step, the clustering algorithm merges the two closest clusters by measuring their distance till, in the last step, only one cluster will remain. All these merging steps are saved in a data structure that can be called *linkage matrix*. Then, another function receives as input this matrix and the number of cluster to generate (k). Finally, returns as output the labels associated to the samples in the starting dataset. There are three parameters that tune the algorithm:

K. It is the number of clusters to generate. It is obtained by the height at which you decide to cut dendogram. To compare hierarchical and k means method, we the same number of cluster, k=5. It is showed by the picture below.



The colors represent the K=5 clusters obtained cutting the dendrogram at the desired height.

Metric. it is the formula to calculate the distance between two points and can influence the final shape of clusters. In our analysis we used the ”Euclidean” metric distance.

Linkage. It measures the distance between two sets of points. The algorithm uses this formula to find what are the closest cluster to merge at a certain moment of execution. In our analysis we used the linkage "complete". The choice of the linkage type in the hierarchical clustering influences the clusters shapes.

We choose the most common linkage that is the complete distance. It is defined as:

$$d(u,v) = max(dist(u[i], v[j]))$$

Where u, v are two clusters and u_i , v_i are two elements in the respective clusters. It is the distance between the farthest pair of points of two different clusters. The algorithm merges the two clusters where this distance lower.

Complete linkage is not strongly affected by outliers, but can break large clusters and has trouble with convex shapes. It is computed in $O(n^2)$.

Thus, the hierarchical clustering algorithm works as follow.

1. Begin with the disjoint clustering having level $L(0) = 0$ and sequence number $m = 0$.
2. Find the max distance pair of clusters in the current clustering.
3. Increments the sequence number $m = m + 1$. Merge clusters u and v into a single cluster to form the next clustering m. Set the level of clustering $to\ L(m) = d[(u),(v)]$.
4. Update the distance matrix, D, by deleting the rows and columns corresponding to clusters (u) and (v) and adding a row and column corresponding to the newly formed cluster. The distance between the new cluster, denoted (u,v) and old cluster (k) is defined in this way: $d[(k),(u,v)] = min\ (d[(k),(u)], d[(k),(v)])$.
5. Repeat the step 2 until all the data points are in one cluster.

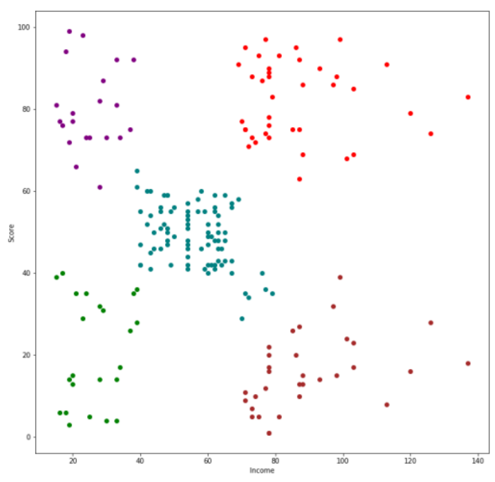
COMPUTATIONAL COSTS: The pseudo code that we used to compute the time complexity is the following.

```
Input:
A set X of object $(x_1, \dots, x_n)$
A distance function $dist(c_1, c_2)$
for i=1 to n
    $c_i = (x_i)$
C=$(c_1, \dots, c_n)$
l=n+1
while C.size > 1 do
    $(c_{min1}, c_{min2}) = min\ dist(c_i, c_j)\ \forall\ c_i, c_j\ in\ C$
    remove $c_{min1}$ and $c_{min2}$ from C
    add $(c_{min1}, c_{min2})$ to C
    l=l+1
```

The standard algorithm for hierarchical agglomerative clustering has a time complexity of $O(n^3)$, and require $O(n^2)$ memory, which makes it too slow for a medium data set. But for some special cases, with complete and single linkage, the algorithm has a the time complexity (n^2) .

SILHOUETTE: We apply the global silhouette score for k=5, in order to compare the result with the k means algorithm. The value that we obtain is a silhouette score=0.5529945955148896.

PLOT:



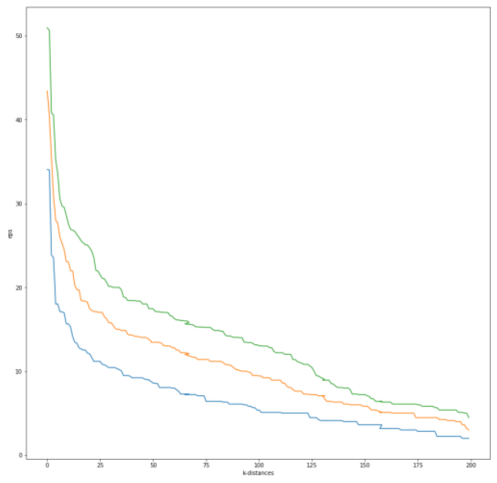
The results is similar to the k-means, since there are 5 well defined different regions, corresponding to the same customers analysed in the k means plotting. Only the blue cluster with "medium" Annual Income and Spending Score seems to be contaminated by some observations. The consideration done above hold also for the hierarchical algorithm.

DBSCAN

DBSCAN works in a different way from the Hierarchical and K means, in fact its biggest difference is that it does not require as configuration parameter the number of clusters (k) that will be generated. The two parameters used to generate labels are *eps* and *min samples*. During the execution phase DBSCAN visits one point at a time tracing an area with ray of *eps* around the point itself. If, in this area there are other points as the value of *min samples*, a new cluster is generated and the point marked as core and all points in the neighborhood are assigned to it, otherwise the visited point is temporarily labelled as noise. In fact, later in the execution, this point could be part of a new cluster because it could be a neighbor of another core point. Note that, when a new cluster is generated, in addition to the core and neighborhood points, also the neighbors of neighborhood points are added to the cluster, this until all points have been visited. Therefore, to correctly configure DBSCAN it is necessary to run several tests by varying the two input parameters *eps* and *min samples*. In the analysis we set *min samples* with heuristic values of 5, 10 and 15 in order to give consistency to the size of the clusters and then we test the value of *eps* and analyzed the results**.**

NEIGHBOR DISTANCE

As we said previously, the DBSCAN algorithm creates clusters by using the points distance. For this reason we adopted a method that allow to calculate the points distances and show them as a function, then it is possible to ease the choose of the configuration parameter *eps* that represents the distance used by the algorithm to generate the clusters labels. The idea is to calculate, the average of the distances of every point to its k nearest neighbors. The value of k will be specified by the user and corresponds to *MinPts*. Next, these k-distances are plotted in an ascending order. The aim is to determine the "knee", which corresponds to the optimal *eps* parameter. A knee corresponds to a threshold where a sharp change occurs along the k-distance curve.



The blue line represents eps for MinPts=5, orange line for MinPts=10, green line for MinPts=15.

Thus, we choose *esp*=10 for *MinPts*=5, *eps*=15 for *MinPts*=10 and *eps*=18 for *MinPts*=15.

COMPUTATIONAL COSTS:

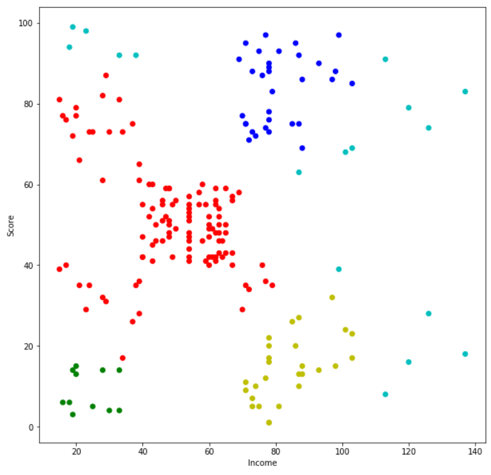
The pseudo code that we used to compute the time complexity is the following.

```
DBSCAN(D,eps,MinPts)
    C=0
    for each P in a data set D do
        if P is visited then
            continue to next point
            mark P as visited
            nbrPts=points in ε-neighborhood of P
        if sizeof(nbrPts)<minPts then
        mark P as NOISE
        else
         C=NewCluster
        Call ExpandClusterFunct(P,nbrPts,C,minPts)
ExpandCluster(P,nbrPts, C, eps, MinPts)
    add P to cluster C
    for each point P' in a nbrPts
        if P' is not visited
        mark P' as visited
        NeighborPts'=points in a $\varepsilon$-neighbor of P'
         if sizeof(nbrPts')>=MinPts
          nbrPts=nbrPts joined nbrPts'
     if P' is not member of any cluster
          add P' to a cluster C
```

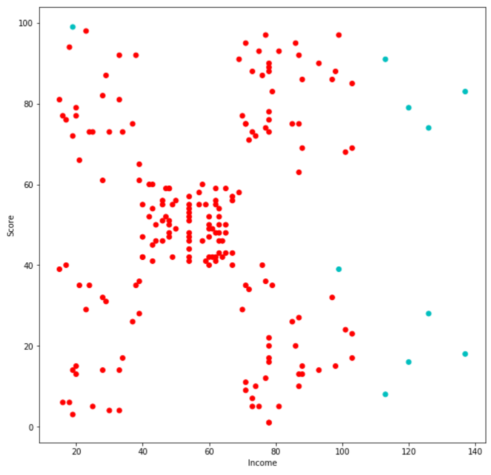
Each query takes $O(n)$ in the worst case, and in the worst case we visit n query. Then the time complexity is $O(n^2)$ while the space complexity is $O(n)$. In some cases, the algorithm can be performed better. If D has been bi-dimensionally sorted, the exact DBSCAN problem (in 2D space) can be solved in $O(n)$ worst-case time. This means that we can even beat the $\Omega(n * logn)$ time bound for this problem in scenarios where sorting can be done fast.

SILHOUETTE: we apply the silhouette method for the pair of *MinPts* and *eps* respectively (5,10): the silhouette score is 0.41249187303464097.

PLOT:



For MinPts=5, eps=10



For MinPts= 10, eps=15

Considering only the first plot, the DBSCAN algorithm divides the data set in four clusters. It easy to see that the red clusters doesn't provide much information about type of customers. It includes both observation with high and low Spending score, i.e. does not capture the propensity to consumption of the customers given their Income. The algorithm is not able to divide the data se in reasonable regions. In contrast to the other two, the DBSCAN is not affected by outliers, because it is able to exclude them from the cluster formation. However, some points labelled as "noise" are not extreme observations of the data set. This algorithm seems not to perform well with this data set, where the

the distribution of the data is globular, and the isolation of the outliers does not seem significatively.

CONCLUSION

From the implementation of several clustering techniques described previously we can say that for this particular data set the k means algorithm is the most suited to classify the data. In fact, the k means is more cost efficient than the hierarchical algorithm having more or less the same silhouette score for the same number of clusters; in this dataset given the globular distribution of the observation the k means outperforms the DBSCAN both in silhouette score and cluster shape.

The main problem about the k means is the random nature of initial centroids, which leads to different clustering results. A possible solution would be performs the k means algorithm several times and choose the best fit according to silhouette score and the shape of the partition.

APPENDIX

In this section we show a simple modification of the k means that yields a significative better result in terms of accuracy and execution time. We already showed that one of the critical points of the k means is the random initial centroids that often converge to a local optimum.

It is possible improve the k means by setting initial centroids based on weighted average. The pseudo-code of this algorithm is the following.

The algorithm calculates a score for each observation, based on the weighted average of its attributes. Then the dataset is sorted by the observation's score, and divide in a partition of k set. After that, you compute centroids.

```
Input:

D=($d_1$,…, $d_n$), the set of n data items

 K number of clusters

Output:

A set of k initial centroids

1. Calculate the average score of each data point
    i.  $d_i$=$x_1$,…, $x_n$;
    ii. $d_i$ ( $avg$ )=( $w_1 * x_1$ +…+ $w_m * x_m$)/$m$, where
        $x$ =the attributes value, $m$ =number of
        attributes and$w$=weight to multiply to ensure
        fair distribution of cluster;

2. Sort the data based on the average score;

3. Divide the data in k subset;

4. Calculate the mean value of each dataset;

5. Take the nearest possible data point of the mean as
   the initial centroid for each data subset.
```

The overall complexity of the proposed algorithm is of $O(n(ki + logn))$. The algorithm shows better performance than standard k means in terms of accuracy and execution time (faster convergence).

REFERENCEES

- K. Jain, R. C. Dubes*, Algorithms for clustering data* (1988)
- D. Defays*, "An efficient algorithm for a complete-link method"* (1977)
- J. A. Hartigan, M. A. Wong "*A K-means clustering algorithm*" (1979)
- S. Mahmud, M.Raham, N. Akhtar "*Improvement of K-means clustering algorithm with better initial centroids based on weighted average*" (2012)

- *https://www.cms.waikato.ac.nz/~abifet/book/chapter_9.html*
- https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/index.html