Ring-like clustering on noisy data.

Castillejo Vela, Manuel

University of Seville

Sevilla, Spain

Mancasvel1@alum.us.es

***Abstract*— The topic of this assignment is inspired on a real subproblem within the LHCb experiment at CERN, although it has been (obviously) significantly simplified. The RICH (Ring Imaging Cherenkov) detectors try to identify patterns with a circular or elliptic patterns which are caused by the impact of Cherenkov radiation on the sensors: cones of photons emitted by particles resulting from hadrons collisions when such particles travel through a special gas at a speed higher than the speed of light in that gas medium. Nevertheless, we will skip all physical details in this assignment, reformulating the problem as follows: Given a collection of points within a given range (for example, a 100x100 square area), the goal is to find the best set of "rings" modelling the given data in such a way that all points from the cloud fit into one of those rings. Of course, we assume that there will be some noise, so that we do not expect a perfect fit, we just try to minimize the error. The term "noise" is used here in a broad sense: it can refer both to "ghost points" that do not belong to any circumference and can be ignored, as well as to small errors in the measurements-(*Abstract*)**

I. INTRODUCTION

The exploration of circular patterns in data analysis holds significant relevance across various domains, including particle physics experiments such as the LHCb experiment at CERN. Within the LHCb experiment, the Ring Imaging Cherenkov (RICH) detectors play a crucial role in identifying circular or elliptic patterns induced by Cherenkov radiation. These patterns emerge as cones of photons emitted by particles resulting from hadron collisions, particularly when these particles traverse a specialized gas medium at velocities surpassing the speed of light within that medium.

While the intricacies of the physical processes involved in the LHCb experiment are complex, our focus here is on an abstraction of a real subproblem encountered within this experiment, albeit significantly simplified for the purposes of this assignment. Specifically, we reformulate the problem in a way more simple way.

In this context, our assignment revolves around the development and evaluation of computational methods capable of robustly detecting circular trajectories amidst noisy data.

To this end I have explored multiple algorithms in order to reach the selected one. Through the research through scientific papers related to the domain of our problem I have tried to reach solutions through HT-based techniques which as many studies have proven it lacks respects memory storage which even with a efficient version of the algorithm it still fails with the AFCS algorithm which is the one that we take into account after this comparison[1].

Some others algorithms that were tried along the process were k-means which lacks with flexibility, is not robust regarding noise and have a significant dismiss due to the representation of the results. C-means, although in the domain of this problem was not allowed to be used the c-means algorithm is more focused on not knowing forms from before and C-shell could be more focused on patterns that were specified from before.

II. MATHEMATICAL FORMULATION

The Fuzzy C-Shell algorithm relies on the concept of fuzzy membership, where each data point belongs to every cluster with a degree of membership ranging from 0 to 1. Let **X={x1,x2,...,xn}** be the dataset consisting of nnn data points in ddd-dimensional space, and **C={c1,c2,...,ck}** be the set of cluster centroids. The degree of membership of data point xi to cluster cj is denoted by uij , where **0≤uij≤1**.

The key equations governing the Fuzzy C-Shell algorithm are as follows:

1. **Distance Calculation:** The distance between two data points x1x_1x1 and x2x_2x2 is computed using various distance metrics such as Euclidean, Manhattan, or ellipsoidal distance. In this case after the only implementation of ring-shaped forms with a constant radius, the Euclidean distance is the one that will be used for the experiments.

   - Euclidean Distance:

$$d_{\text{euclidean}}(x_1, x_2) = \sqrt{\sum_{i=1}^{d}(x_{1i} - x_{2i})^2}$$

- Manhattan Distance:

$$d_{\text{manhattan}}(x_1, x_2) = \sum_{i=1}^{d} |x_{1i} - x_{2i}|$$

Ellipsoidal Distance: As in the paper of Adaptative Fuzzy c-Shells clustering the ellipsoidal distance would be the chosen in order to perform the classification in the clustering.
In the domain of our problem the ellipses are not included by the time our main focus is on perfect circular shapes. Although the distance implementation is completed and could be added in future implementations, in this experiment I overlook the the correct addition of parametres a and b in addition of all the experiments that it entails.

$$d_{\text{elipsoidal}}(x_1, x_2) = \sqrt{\left(\frac{x_{1x}-x_{2x}}{a}\right)^2 + \left(\frac{x_{1y}-x_{2y}}{b}\right)^2}$$

2. **Membership Matrix Initialization:** Before calculating the centroids and radii of the clusters, it's necessary to initialize the membership matrix U. This matrix U has dimensions k×nk \times nk×n, where k is the number of clusters and nnn is the number of data points in the dataset. Each element uij_uij of the matrix represents the membership of point xix_ixi to cluster cjc_jcj. In initialization, random values are assigned to the elements of U and then normalized so that the sum of memberships of each point to all clusters equals 1.
3. **Centroid and Radius Computation:** Once the matrix has been initialized the centroid are defined as points in the feature space that represent the geometric center of the data assigned to each cluster.

$$c_j = \frac{\sum_{i=1}^{n} (u_{ij})^m \cdot x_i}{\sum_{i=1}^{n} (u_{ij})^m}$$

Where cj is the centroid of cluster j and m is the fuzziness parameter.

Once we have the centroid is calculated the next step is the radii of the cluster in order to calculate the dispersion of the data around the centroids. It represents the the average distance of the points assigned to each cluster to their respective centroid.

$$r_j = \sqrt{\frac{\sum_{i=1}^{n} (u_{ij})^m \cdot d(x_i, c_j)^2}{\sum_{i=1}^{n} (u_{ij})^m}}$$

Where rj is the radius oof cluster j and d(xi,cj) the distance between point xi and centroid cj.

4. **Membership Matrix Update:** The membership matrix is updated iteratively using the calculated centroids and radii. The degree of membership of each data point to each cluster is adjusted based on the distances to cluster centroids and the defined fuzziness parameter m.

$$I_k = \phi \Rightarrow u_{ik} = \frac{1}{\sum_{j=1}^{n} \left[\frac{(D_{ik})^2}{(D_{jk})^2}\right]^{\frac{1}{(m-1)}}}$$

5. **Optimal Cluster Number Determination:** As an extra addition the optimal number of clusters is determined using the elbow method, which identifies the point of maximum curvature in the sum of squared errors (SSE) plot.

$$\text{Optimal number of clusters} = \arg\min\left(\frac{d^2(\text{SSE})}{d^2k}\right) + 2$$

.

III. Experimentation and Conclusions

The study, we conducted a series of experiments to evaluate the performance and robustness of the Fuzzy C-Shell clustering algorithm. The experiments were carried out on synthetically generated data, specifically designed to test the algorithm's capability to handle both structured clusters and random noise. Below, we detail the experimental setup, the modifications made during our iterative process, and the outcomes observed.

a) Initial Data Generation

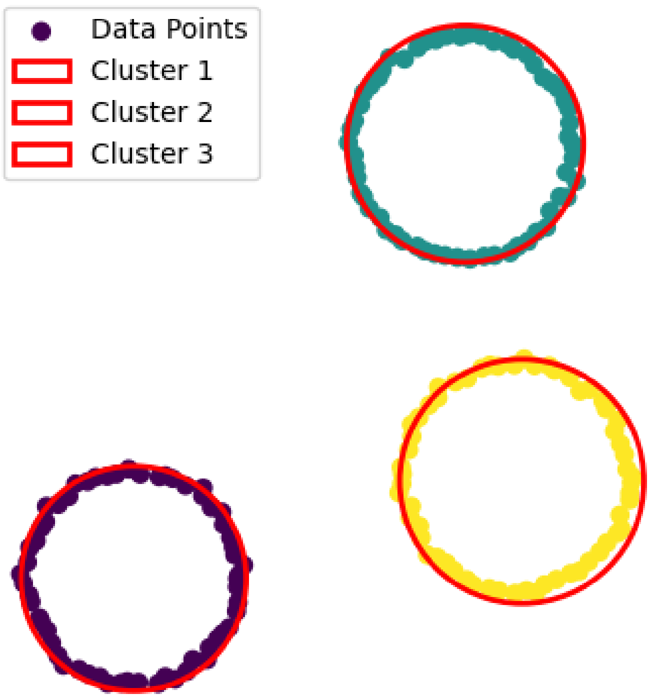
The foundation of our experiments involved generating synthetic datasets using the following method:

1. **Circular Clusters**: We generated multiple circular clusters, each characterized by:
    - **Number of Circles**: Three distinct circular clusters were the main focus by the time with the generation could intersect or not in the space we have in control. With more rings they will always intersect or the probabilities to not intersect are smaller
    - **Points per Circle**: Variety of points points uniformly distributed around each circle.
    - **Cluster Centers and Radii**: Centers were randomly selected within a 2D space, and radii were varied to create circles of different sizes.
    - **Displacement**: Small random displacements were added to points to simulate natural variability and avoid perfect circular shapes.
2. **Noise Points**: To simulate real-world data imperfections, we included:
    - **Number of Noise Points**: From zero to ten points and 50 points scattered randomly across the dataset.

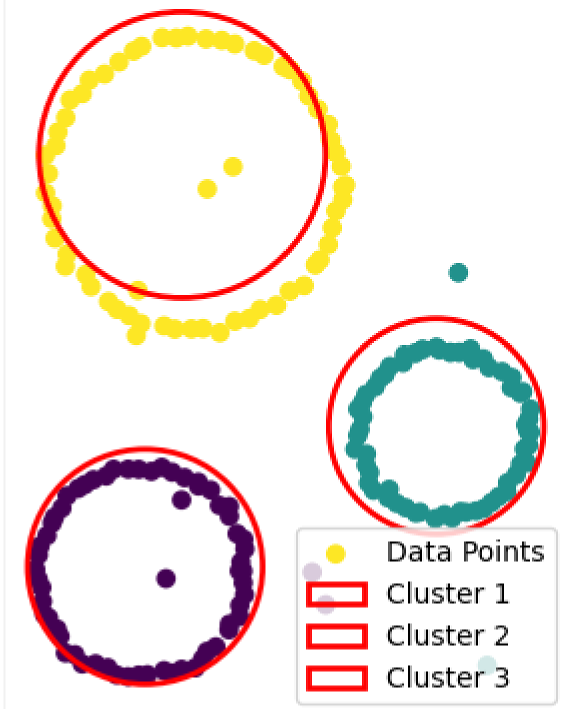
Identify applicable funding agency here. If none, delete this text box.

- **Noise Distribution**: Noise points were uniformly distributed within the same space as the clusters in order to create that uncertainty.

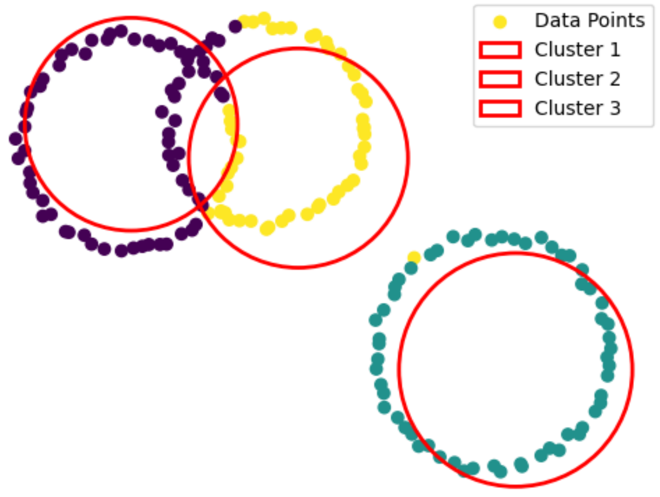
The data was then saved in CSV format for subsequent use in clustering experiments.



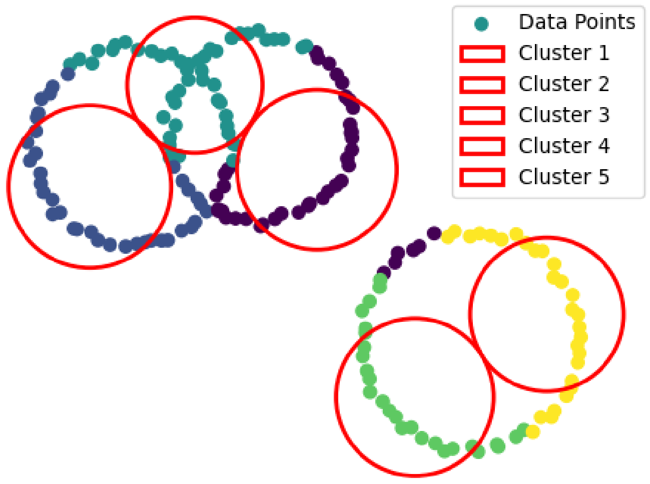
Results with variance between circle points and no noise.



Results with variance between circle points and noise.



Results between circle points and circles intersections with a limited number of rings below 10.



Results between circle points and circles intersections with a limited number of rings to 10 or more.

As we can see the algorithm performs well in conditions of variance between circle points without noise. The performance goes quite down when some noise it is added but the rings detection is quite identical.

The main problem of this algorithm is the intersection. When two generated circles intersect the algorithm could detect almost the circles that are part of the intersection if the number of the maximum cluster is limited to a number below 10. Otherwise, the algorithm will detect the points of intersection as another different circle.

The main reason for this event is that my algorithm is not capable of detecting partial circles and represent them as partial circles or ellipses so when two circles intersect between both of them a ellipse is formed which it could be detected and 2 semicircles are detected along that central ellipse.

IV. Elbow Method for Determining the Optimal Number of Clusters

The Elbow Method is a popular technique used to determine the optimal number of clusters in a dataset when using clustering algorithms such as K-means or Fuzzy C-Shell. The method involves plotting the Sum of Squared Errors (SSE) for different numbers of clusters and selecting the point where the SSE starts to decrease less sharply, resembling an "elbow".[2]

a) What is the Elbow Method?

The Elbow Method is based on the principle that as the number of clusters increases, the SSE (also known as the inertia) will decrease because the clusters become smaller, and their centroids are closer to the data points. However, after a certain point, adding more clusters does not significantly reduce the SSE, indicating that the optimal number of clusters has been reached. This point is known as the "elbow."

b) Mathematical Explanation

$$\mathrm{SSE} = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

- k is the number of clusters.
- Ci is the set of points in cluster iii.
- μi is the centroid of cluster iii.
- x is a data point.
- ∥x−μi∥^2 is the squared Euclidean distance between a data point and the centroid of its cluster.

As the number of clusters increases, the SSE will decrease. The goal is to identify the number of clusters k at which adding an additional cluster does not provide a significant improvement in the SSE.

c) Implementation

We implemented the Elbow Method as part of our Fuzzy C-Shell clustering algorithm to determine the optimal number of clusters. Here is a detailed explanation of the implementation steps[3]:

1. **Initialization**: We begin by fitting the Fuzzy C-Shell model to the data for a range of cluster numbers, from a minimum of 2 to a maximum specified by the user but limited to 100.
2. **Compute SSE for Each Cluster Count**: For each number of clusters, we calculate the SSE using the membership matrix and centroids computed by the algorithm.
3. **Plot SSE vs. Number of Clusters**: We plot the SSE values against the corresponding number of clusters to visualize the decrease in SSE as the number of clusters increases.
4. **Identify the Elbow Point**: The optimal number of clusters is determined by identifying the "elbow" point on the plot, where the rate of decrease in SSE sharply reduces.

d) Improvements Made

Enhanced Gaussian Smoothing

Gaussian smoothing is applied to the SSE curve to reduce noise and make the elbow point more prominent. The Gaussian smoothing function is given by:

$$S\hat{S}E(x) = \sum_{i=-n}^{n} SSE(x+i) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{i^2}{2\sigma^2}}$$

where:

- SSE^(x)is the smoothed SSE at point x.
- σ is the standard deviation of the Gaussian kernel.
- n is the range of the kernel.

Maximum Curvature Detection

To automatically detect the elbow point, we calculate the curvature of the SSE curve. The point of maximum curvature corresponds to the elbow. The curvature κ at a point x on a 2D curve (x,SSE(x)) can be calculated using:

$$\kappa = \frac{|y''|}{(1+(y')^2)^{3/2}}$$

where:

- y′ is the first derivative of the SSE curve.
- y′′ is the second derivative of the SSE curve.

V. Fuzzy implementation

In the Fuzzy C-Shell clustering algorithm, the concept of fuzzy membership plays a central role, allowing for a more nuanced representation of data points' association with clusters. This section elaborates on the mathematical formulation and implementation of fuzzy membership within the algorithm, particularly focusing on the _update_membership_matrix method.

a) Mathematical Formulation

The _update_membership_matrix method is responsible for recalculating the membership matrix, which quantifies the degree to which each data point belongs to each cluster. Given N data points and K clusters, the membership matrix U is an K×N matrix, where each element uij represents the membership degree of the i-th data point to the j-th cluster. The degree of membership uij is calculated based on the distances between data points and cluster centroids. Specifically, it is computed using a fuzzy membership function that takes into account the distance between a data point and all centroids. The parameter mmm, which ranges between 1 and infinity, controls the fuzziness of the clustering, with larger values of mmm leading to more pronounced fuzzy membership[4].

b) Implementation Details

The _update_membership_matrix method iterates over each data point and centroid pair, computing the degree of membership using the following steps:

1. Compute the distance between the data point and each centroid.
2. Apply the fuzzy membership function, which involves calculating the ratio of the squared distances and taking the reciprocal raised to the power of (1)/(m – 1) .
3. Normalize the membership values to ensure that they sum up to one across all clusters for each data point.

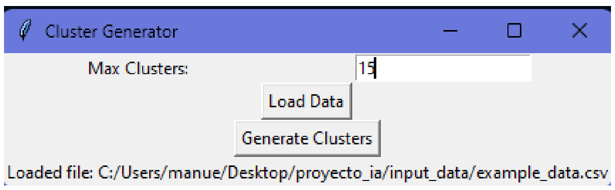
This process results in an updated membership matrix that reflects the updated degree of association between each data point and cluster, considering both the distances and the fuzziness parameter m.

c) Discussion

The incorporation of fuzzy membership in the Fuzzy C-Shell algorithm enables a more flexible and nuanced representation of the clustering structure within the data. By allowing data points to belong to multiple clusters simultaneously with varying degrees of membership, the algorithm can capture complex data relationships and provide richer insights into the underlying structure of the dataset.

VI. Extra implementations and future updates

Right now the code have very simple visual interface in which the user could only modify the maximum number of clusters (between some certain numbers), and a button which leads the user to the archives explorer of their pc in order to load the data in which the algorithm will perform.



Max Clusters: 15

Loaded file: C:/Users/manue/Desktop/proyecto_ia/input_data/example_data.csv

Visualization of the actual interface.

The implementation that could be the more important in order to perform in all cases could be the detection of partial circles and adapt the algorithm to intersections and ellipses.

Some other more advance implementations that could be added in the future could be a field in which the user could select which type of figure he wanted to find and the algorithm will find the figures in the data given.

Another addition could be the improvement of the visual interface with the customization of the background, fields and buttons.

VII. Conclusion

This document presents a detailed exploration and implementation of the Fuzzy C-Shell clustering algorithm to address the problem of identifying ring-shaped patterns within noisy data, inspired by the real-world challenges faced in the LHCb experiment at CERN but in a more simplified manner. The algorithm's ability to handle fuzzy memberships and incorporate flexible cluster shapes provides a robust solution for modeling ring-shaped data patterns.

a) Key Findings

1. **Fuzzy Membership and Clustering**: The Fuzzy C-Shell algorithm employs a unique approach by allowing data points to belong to multiple clusters with varying degrees of membership, denoted as uij. This flexibility enables the algorithm to handle complex data distributions more effectively than traditional hard clustering methods.
2. **Improved Elbow Method**: To determine the optimal number of clusters, we enhanced the traditional Elbow Method by incorporating Gaussian smoothing and maximum curvature detection. This approach provided a more accurate identification of the "elbow" point, ensuring the selection of an appropriate number of clusters for various datasets.
3. **Synthetic Data Experiments**: Extensive experiments were conducted using synthetically

generated datasets with different configurations, from separated rings, to rings that intersect and the addition or deletion of noise in the data compiled. The results demonstrated the algorithm's robustness in identifying circular patterns even in the presence of significant noise. However, challenges were observed when handling intersecting circles, indicating areas for future improvement.

b) Contributions and Future Work

This study contributes to the field of clustering by providing an implementation and evaluation of the Fuzzy C-Shell algorithm, highlighting its advantages and limitations. The incorporation of fuzzy memberships and the enhancements to the Elbow Method to detect the number of clusters without any input.

Future work can focus on addressing the limitations observed with intersecting circles by incorporating techniques for detecting partial circles and ellipses. Additionally, further optimization of the algorithm's performance and scalability could enhance its applicability to larger and more complex datasets.

VIII. References

This whole document was tailored with the information from different papers, websites and artificial intelligence which was used in order to find papers with related information about the topic we are treating and to achieve a better structure and comprehension about the subject that has been presented.

[1] Davé, R. N., & Bhaswan, K. (1992). Adaptive fuzzy c-shells clustering and detection of ellipses. IEEE Transactions on Fuzzy Systems, 3(3), 643-652.

[2] "K-Means Clustering Optimization Using the Elbow Method and Early Centroid Determination Based on Mean and Median Formula" (ResearchGate, 2019)

[3] "Coding to plot the Elbow Curve for the Elbow method"

[4] Fuzzy clustering analysis for optimizing fuzzy membership functions

Mu-Song Chen, Shinn-Wen Wang