# Clustering Rings in Noisy Data

1st David González Martínez
*University of Seville*
Seville, Spain
davgonmar2@alum.us.es

***Abstract*—In this paper, we apply the Fuzzy K-Rings algorithm, also known as the Fuzzy C-Shells algorithm, to ring clustering in noisy data. We further propose a modification to the algorithm to make it more robust to noise, and conduct different experiments to test the performance of the algorithm.**

***Index Terms*—Fuzzy K-Rings, Fuzzy C-Shells, Clustering, Noisy Data, Ring Clustering, Hypersphere Clustering, Noisy Rings Clustering**

## I. Introduction

We are presented with the following problem: we have a dataset that is composed of different rings and noise. Our objective is to classify the points into different clusters, each corresponding to a different ring. Fuzzy clustering algorithms, instead of assigning a single cluster to each data point, assign a membership degree to each data point for each cluster. There has been a significant amount of work in the past on fuzzy algorithms applied to different types of datasets. The Fuzzy K-Rings algorithm, also known as the Fuzzy C-Shells algorithm, is a clustering algorithm that has been used in the past for similar tasks. The algorithm is inspired by the Fuzzy C-Means algorithm, and was introduced, although in different variations, in [1] and [2]. Other papers that make use of different versions of this algorithm are [3] and [4]. In this paper, we will focus on applying the Fuzzy K-Rings algorithm to the problem of clustering rings in noisy dataset. By noise we mean inconsistency in the rings, or background noise that does not belong to any ring. We will further propose a modification to the algorithm to make it more robust to noise, and conduct various experiments to test the performance of the algorithm under different circumstances.

## II. Fuzzy Clustering Algorithms

In hard clustering algorithms, each data point is assigned to a single cluster. In contrast, fuzzy algorithms assign a membership degree to each data point for each cluster. One of the most popular ones is the Fuzzy C-Means algorithm. A formal description of it can be found in [5] and [6]. The algorithm can be seen as an optimization problem, where the objective function is to minimize the following equation:

$$J(U,V) = \sum_{i=1}^{N}\sum_{j=1}^{K}(u_{ij})^q(d_{ij})^2 \tag{1}$$

where $K$ is the number of clusters, $N$ is the number of data samples, $u_{ij}$ is the membership degree of cluster $i$ to data sample $j$, $d_{ij}$ is the euclidean distance between data point $i$

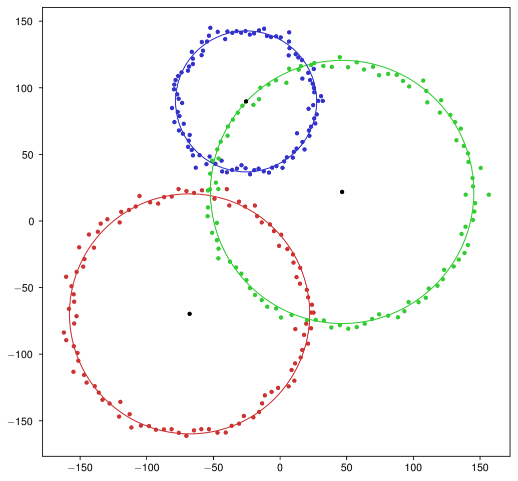

Fig. 1. Example of a dataset with different rings and noise, and their correct classification.

and the center of cluster $j$, and $q$ is a parameter in $[1, \infty)$ that controls the fuzziness of the membership degrees. The higher the value of $q$, the 'fuzzier' the algorithm will be. When $q$ is 1, the algorithm will be equivalent to K-Means, that is, it will be a hard clustering algorithm. So, if we focus strictly on fuzzy algorithms, then $q$ is in $(1, \infty)$, and if we include hard clustering algorithms, then $q$ is in $[1, \infty)$. $U$ is a matrix of size $n \times k$, and can be interpreted as 'how much data point $i$ belongs to cluster $j$'. It is important to note that the following conditions must be met, as described in [6]:

1) $u_{ij} \in [0, 1]$
2) $\sum_{j=1}^{K} u_{ij} = 1$

## III. The Fuzzy K-Rings Algorithm

The Fuzzy K-Rings algorithm is a clustering algorithm that is able to cluster data points in a ring-shaped dataset. The algorithm is inspired by the Fuzzy C-Means, and described in [2] and [1], altough in different variations. It is described as an optimization problem, where the objective function is to minimize the following equation:

$$J_q(U,V,R) = \sum_{i=1}^{N}\sum_{j=1}^{K}(u_{ij})^q(d_{ij} - r_i)^2 \tag{2}$$

where $K$ is the number of rings, $N$ is the number of data samples, $u_{ij}$ is the membership degree of cluster $i$ to data sample $j$, $d_{ij}$ is the euclidean distance between data point $i$ and the center of cluster $j$, $r_i$ is the radius of the cluster $i$, and $q$ is a parameter that controls the fuzziness of the membership degrees. From now on, we'll refer to

$$|d_{ij} - r_i|$$

as $d'_{ij}$. Now, we'll describe the ways to update the different parameters in the algorithm, and then we'll describe the initialization and convergence criteria, as well as the concrete steps.

### *A. Updating the Membership Degrees*

The membership degrees are updated using the following equation, as described in both [2] and [1]. It's the same as the one used in the Fuzzy C-Means algorithm, but with $d_{ij}$ replaced by $d'_{ij}$.

$$u_{ij} = \frac{1}{\sum_{k=1}^{K} \left(\frac{d'_{ij}}{d'_{ik}}\right)^{\frac{2}{q-1}}} \tag{3}$$

### *B. Updating the Cluster Radii and Centers*

As mentioned, we can define the algoritm as an optimization problem after fixing $U$. We can then obtain the optimal (minimum) values for the objective function by setting the partial derivatives with respect to $r_i$ and $V_i$ to zero. First, we have:

$$\frac{\partial}{\partial r_i}(J_q) = \sum_{j=1}^{N} (u_{ij})^q \frac{\partial}{\partial r_i}(d_{ij} - r_i)^2 = \sum_{j=1}^{N} (u_{ij})^q (r_i - d_{ij}) = 0 \tag{4}$$

For the centers, we take a different approach to the one taken in [1], and similar to the one in [2]. It is more computation friendly, and it can be trivially extended to higher dimensions, unlike [1].

Let $X_j$ be a data point, and $V_i$ be the center of cluster $i$. Let $d_{ij}$ be the euclidean distance between $X_j$ and $V_i$, and $r_i$ be the radius of cluster $i$. Let $d'_{ij}$ be the distance between $X_j$ and the circle with center $V_i$ and radius $r_i$.

Then, let the following be true:

$$V'_i = \frac{r_i}{d_{ij}} X_j + (1 - \frac{r_i}{d_{ij}}) V_i \tag{5}$$

Differentiating (2) with respect to $V_i$ and setting it to zero, we get:

$$\frac{\partial}{\partial V_i}(J_q) = \sum_{j=1}^{N} u_{ij}^q \frac{\partial}{\partial V_i}(d'_{ij})^2 = 0 \tag{6}$$

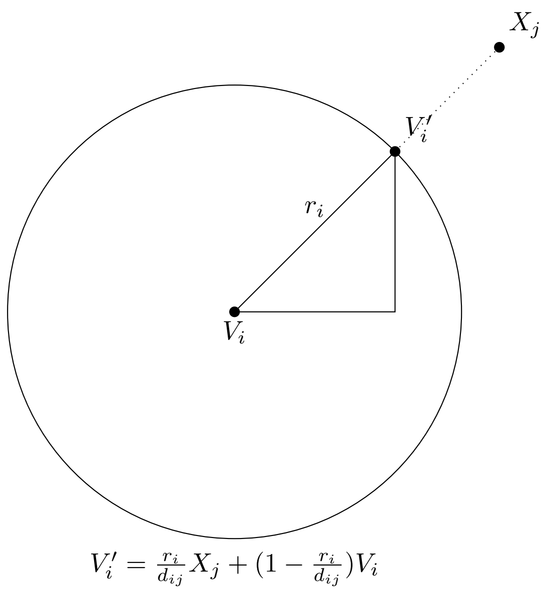
$X_j$
$V'_i$
$r_i$
$V_i$
$V'_i = \frac{r_i}{d_{ij}} X_j + (1 - \frac{r_i}{d_{ij}}) V_i$

Fig. 2. Visualization of the geometrical meaning of the update of the cluster centers.

Note that we can rewrite $(d'_{ij})^2$ as $(X_j - V'_i)^T(X_j - V'_i)$. Then, following [1], we can solve:

$$\begin{aligned}
\frac{\partial}{\partial V_i}(d_{ij} - r_i)^2 &= \left(\frac{\partial}{\partial V_i}(|X_j - V_i| - r_i)^2\right) \\
&= -2\left((X_j - V_i) - \frac{r_i}{|X_j - V_i|}(X_j - V_i)\right) \\
&= -2\left((X_j - V_i) - \frac{r_i}{d_{ij}}(X_j - V_i)\right) \\
&= -2\left((1 - \frac{r_i}{d_{ij}})X_j - (1 - \frac{r_i}{d_{ij}})V_i\right)
\end{aligned} \tag{7}$$

Plugging that into (6), we get:

$$\sum_{j=1}^{N} u_{ij}^q (1 - \frac{r_i}{d_{ij}})(X_j - V_i) = 0 \tag{8}$$

We now have a system of equations that we can solve for $V_i$ and $r_i$ to obtain the critical points of the objective function. It is important that since the equations are coupled, they must be solved together. [2] mentions (and cites the proof) that, indeed, the critical points are minima. The experimental results also back this up. One solution, as noted in [2], is:

$$V_i = \frac{\sum_{j=1}^{N} u_{ij}^q X_j}{\sum_{j=1}^{N} u_{ij}^q} \tag{9}$$

$$r_i = \frac{\sum_{j=1}^{N} u_{ij}^q d_{ij}}{\sum_{j=1}^{N} u_{ij}^q} \tag{10}$$

On the other hand, [1] proposes a different solution, which is to solve the equations separately. In our experiments, we found that solution not to work very well in practice, and the one proposed by [2] to work better.

## C. Intuition

After having given a formal description of the basic algorithm, we can give an intuitive explanation of it.

First, for the membership degrees, we can see that the algorithm is trying to assign higher membership degrees to points that are closer to the ring contour. This is done by averaging the distances of the points to the different rings, and assigning them proportionally.

As for the cluster centers, the algorithm is just using a weighted average of the points, with the weights being the membership degrees. This is similar to the K-Means algorithm, but with the weights being the membership degrees instead of a binary value, and exactly the same as the Fuzzy C-Means algorithm.

Finally, for the radii, the algorithm is trying to assign the radii by using a weighted average of the distances of the points to the cluster centers. This is done by using the membership degrees as weights, and the distances as the values to be averaged.

## D. Initializing the Parameters

[1] proposes two initialization methods, depending on the nature of the data. We adopt both:

- Concentric datasets: In this case, since all rings share the same center, but have different radii, the procedure is as follows. For the centers, we simply compute the baricenter of the dataset:

$$V_i = \frac{1}{n}\sum_{j=1}^{N} X_j \tag{11}$$

Then, for the radii we define max and min as follows:

$$r_{\text{max}} = \max_j d(X_j, V_i) \tag{12}$$

$$r_{\text{min}} = \min_j d(X_j, V_i) \tag{13}$$

They denote the maximum and minimum distance of a point to the center. Then, we can initialize the radii as sampling from a uniform distribution:

$$r_i = r_{\text{min}} + (r_{\text{max}} - r_{\text{min}}) \cdot \text{rand}() \tag{14}$$

$rand()$ is a random number from a uniform distribution in $[0, 1]$.

- Non-concentric datasets: In this case, the rings do not share the same center. The rings can also interlock, and their radii are usually different. As described in [1], we first run the Fuzzy C-Means algorithm on the dataset. They proposed to run only 3 iterations. However, we run the algorithm until convergence. After that, we directly use the membership degrees and centers from the FCM results as our initial status. As for the radius, we obtain it with equation (10).

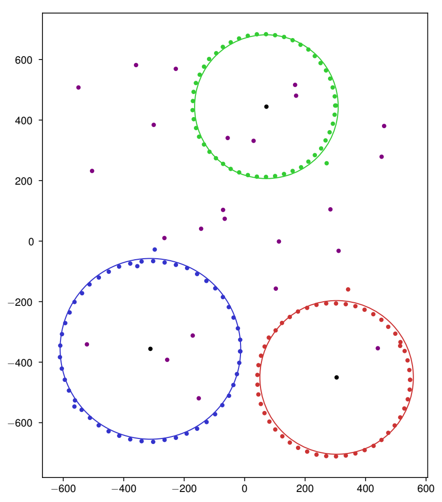

Fig. 3. Example of a dataset with different rings and background noise, and their correct classification (purple means noise).

## E. Convergence Criterion

We use the following convergence criterion:

$$|\hat{u_{ij}} - u_{ij}| < \epsilon \quad \forall i, j \tag{15}$$

Where $\hat{u_{ij}}$ is the membership degree of the previous iteration, and $u_{ij}$ is the membership degree of the current iteration, and $\epsilon$ is a small value, usually $10^{-3}$, given as a hyperparameter. That is, after each update, we check for the difference between the membership degrees of the current and previous iteration, and if the difference is smaller than $\epsilon$, we break the loop. However, we do not stop it completely, but we will get to that in the next section.

## F. Background noise detection

Recall that our objective is to explore the clustering of rings in noisy data. However, in the case of noisy data, the basic algorithm can be sensitive to noise. To mitigate this, we propose an aditional step. The algorithm takes an additional noise threshold as hyperparameter, which is the maximum distance a point can have to all cluster centers to be considered noise. We can express the equation as:

$$\text{is\_noise}(X_j) = \begin{cases} 1 & \text{if } \min_i d_{ij} > \text{noise\_distance\_threshold} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

When computing the centers and radii, points that are considered noise are ignored, that is, all their weights are set to zero. It is important to note that it is only for the computation of the centers and radii. The stored membership degrees are not modified. This is done with the use of a mask, where 1 means not noise, and 0 means noise, and then by multiplying the mask by the membership degrees and the distances. Therefore, the equations to update the centers and radii are modified as follows:

$$V_i = \frac{\sum_{j=1}^{N} u_{ij}^q X_j \cdot mask_j}{\sum_{j=1}^{N} u_{ij}^q \cdot mask_j} \tag{17}$$

$$r_i = \frac{\sum_{j=1}^{N} u_{ij}^q d_{ij} \cdot mask_j}{\sum_{j=1}^{N} u_{ij}^q \cdot mask_j} \tag{18}$$

This makes 'noise' points not to affect the computation of the centers and radii, and therefore, the algorithm is more robust to noise, as shown in the experiments. The recomputation of the membership matrix is not altered. The mask can be obtained by iterating over the data samples, and checking if the noise condition holds:

$$\text{mask}_j = \text{not}(\text{is\_noise}(X_j)) \tag{19}$$

We use a logical not since we want points set to 1 to be not noise, and points set to 0 to be noise. This is so we can use the mask efficiently in the equations.

## G. Tying it all together

Given the mathematical details, we can define the following hyperparameters:

- q: The fuzziness parameter. It controls how 'fuzzy' the membership degrees are.
- convergence_eps: The convergence criterion value. It is usually set to a low value, like $10^{-5}$.
- max_iters: The maximum number of iterations.
- noise_distance_threshold: The maximum distance a point can have to a cluster.
- max_noise_checks: The maximum number of times the noise mask can be recomputed.
- apply_noise_removal: A boolean that indicates if the noise removal step should be applied.
- init_method: The initialization method. It can be either "concentric" or "fuzzycmeans".

All those parameters can be related to the methodology described in the previous sections. The main loop of the algorithm can be seen as follows:

```
let U, V, R = initialize()
let iter = 0
let noise_checks = 0
let noise_mask = ones(n)
let last_noise_mask = zeros(n)

while iter < max_iters:
    U = update_membership()
    R,V = update_radii(),update_centers()
    if convergence_criterion():
        if not apply_noise_removal:
            end()
        noise_mask = get_noise_mask()
        if noise_mask == last_noise_mask:
            end()
        else:
            last_noise_mask = noise_mask
            noise_checks += 1
            continue()
    else:
        continue()
```

Note that the centers and radii should be updated at the same time, since they are coupled equations. That is, we need to compute both before changing any of them. After calling end(), we would obtain the results. Note that this is simplified pseudocode. For a more complete version, see the actual implementation.

## H. Obtaining the results

After the algorithm has converged or we have reached the maximum number of iterations, we can obtain the results. Recall that the membership degree can be seen as 'how much a point belongs to a cluster', and each vector can be seen as a probability distribution. Having this in mind, there are multiple ways we could obtain the results:

1) We can assign each point to the cluster with the highest membership degree.
2) We can sample from a multinomial distribution with the membership degrees as the probabilities.
3) We can simply use the membership degrees directly.

We chose the first option, that is, assigning each point to the cluster with the highest membership degree. As for the radius and the center, obtaining them is a direct result of the algorithm.

## I. Memory Complexity

First, it is important to note that the actual memory complexity and the time complexity, depends on the implementation. For example, by using Python tensor frameworks, such as NumPy [7] or PyTorch [8], we can play with dimension broadcasting and vectorization to make the algorithm faster. Therefore, we only focus on memory complexity, and we give a rough estimate.. First, to store the temporary variables, the memory complexity is the following (k denotes clusters, n number of samples, and d the dimensionality of the data (usually 2)):

1) U: $O(k \cdot n)$
2) V: $O(k \cdot d)$
3) R: $O(k)$
4) mask: $O(n)$
5) last_mask: $O(n)$
6) X: $O(n \cdot d)$

We do not consider temporary variables, such as product of computations, since that is highly dependent on the implementation. We do not take into consideration time complexity, because that, again, is highly dependent on the implementation. The actual memory requirements depend on the precision used for the different tensors. In our case, we found that using 32-bit floats got good results. In the past implementation, we tried using 64-bit floats, but the algorithm was slower, and the results were not better.

## IV. Experiments

We conducted different experiments to test the performance of the algorithm. Given a dataset, that is, a set of points, we could define, informally, two types of points, those that belong to a ring, and those that don't belong to any. We call the second one 'background noise' from now on. In order to generate the dataset, we generate N rings with n noise. The noise of the rings can be seen as 'imperfection' that would occur in a real dataset, for example, because of human error or measurement error. Moreover, we generate $N$ noise points randomly accross the space.

### A. Evaluation Metrics

We use the following metrics to evaluate the performance of the algorithm:

- Absolute distance error (with hard labels)

$$\text{ADE} = \sum_{i=1}^{N} distance(X_i, ring_i) \tag{20}$$

  Where $distance(X_i, ring_i)$ is the distance between the point $X_i$ and the circunference of the ring $ring_i$. $ring_i$ denotes the classified ring of the point $X_i$. Recall that we could classify some points as noise. In the case of a noise point, distance returns 0, that is, we do not take noise points into account when computing the ADE. It is important to mention that this metric has a flaw. The algorithm could simply classify every point as noise, and the error would be minimized. We did not (visually) detect that behaviour in the experiments, but we still include the number of detected noise points in the results (as well as the intended noise points).

On the other hand, we are also interested in getting the lowest runtime possible. We measure the runtime of the algorithm (in seconds), as well as the total number of iterations it takes to converge. We also track the number of detected noise points, and the number of intended (since the datasets are artificially generated) noise points.

### B. Results

We conducted different experiments to test the performance of the algorithm.

*1) General test with excentric rings:* We performed a general test with excentric rings, with different levels of noise and different numbers of rings. The hyperparameters were set as follows:

- q: 1.1
- convergence_eps: $10^{-5}$
- max_iters: 10000
- noise_distance_threshold: 100
- max_noise_checks: 20
- apply_noise_removal: True
- init_method: "fuzzycmeans"

Each circle had 100 samples. The data was generated in the following way:

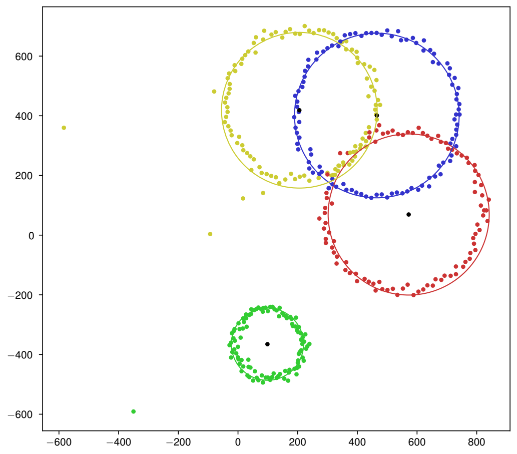

Fig. 4. Example of a dataset with 4 noisy rings, and background noise, and a good classification.

- For each ring, we randomly select a center in a rect centered at $(0, 0)$ with sides of length 1200.
- Each circle had a radius between 100 and 400.
- For each ring, we randomly select 100 samples in the circunference. For each sample, we add noise with the following equation:

$$X_{\text{noise}} = X_{\text{ring}} + \text{randn}(0, 1) \cdot \text{noise\_level} \tag{21}$$

  Where $X_{\text{ring}}$ is the set of points in the circunference, and noise_level is the noise level, and $\text{randn}(0, 1)$ is a random vector from a normal distribution with mean 0 and variance 1.
- To add the background noise, we select N points in the rect, sampled from an uniform distribution.

It is noteworthy to say that the algorithm is sensible to the different hyperparametrs. As we can see in the results, both the runtime and performance degrades with higher noise and higher number of rings.

*2) Concentric Rings:* A concentric ring dataset is a dataset in which all the rings share the same center, and may vary in radius.

The hyperparameters were set as follows:

- q: 1.1
- convergence_eps: $10^{-5}$
- max_iters: 10000
- noise_distance_threshold: 100
- max_noise_checks: 20
- apply_noise_removal: True
- init_method: "concentric"

For the data generation:

- For each ring, we randomly select a radius between 50 and 1000.

| Number of rings | Ring noise | Background noise | Avg. Error | Avg. Runtime | Iterations | Experiments | Avg. Detected Noise |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1.83853 | 0.0007633 | 2 | 3 | 0 |
| 1 | 20 | 0 | 16.2363 | 0.000687367 | 2 | 3 | 0 |
| 1 | 20 | 20 | 16.1469 | 0.00156157 | 6 | 3 | 18 |
| 2 | 0 | 0 | 1.62186 | 0.456976 | 2505.25 | 4 | 0 |
| 2 | 10 | 0 | 14.0226 | 0.00330033 | 10.6667 | 3 | 0 |
| 2 | 20 | 0 | 16.0351 | 0.631052 | 3344 | 3 | 0 |
| 2 | 10 | 10 | 13.5069 | 0.634505 | 3341.33 | 3 | 3.33333 |
| 3 | 0 | 0 | 67.6204 | 2.95479 | 10000 | 5 | 8 |
| 3 | 10 | 0 | 14.4416 | 0.763798 | 2525.25 | 4 | 10.75 |
| 3 | 10 | 15 | 17.266 | 2.04509 | 6670.67 | 3 | 3.33333 |
| 4 | 0 | 0 | 26.2515 | 3.58296 | 8003.8 | 5 | 0 |
| 4 | 10 | 0 | 15.6864 | 1.15821 | 2520.5 | 4 | 5.5 |
| 4 | 10 | 10 | 22.5589 | 4.47653 | 10000 | 3 | 30.6667 |
| 5 | 0 | 0 | 20.2154 | 6.43739 | 10000 | 4 | 0 |
| 5 | 10 | 0 | 14.6675 | 6.43317 | 10000 | 2 | 0 |
| 5 | 10 | 10 | 33.6885 | 6.58792 | 10000 | 3 | 0 |

Fig. 5. Results of the general test with excentric rings. 'Experiments' denote the total number of experiments conducted with the same parameters.

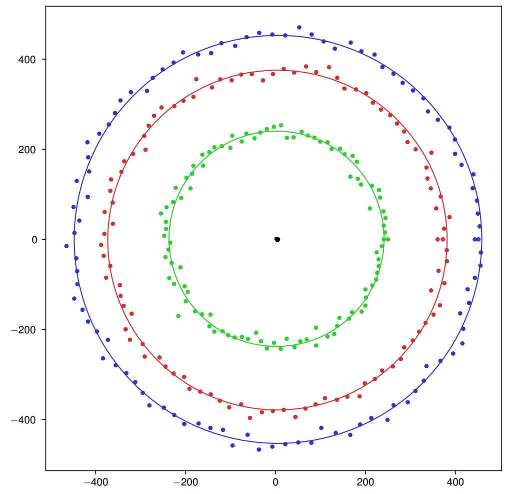

Fig. 6. Example of a dataset with 4 concentric rings, and background noise, and a good classification.

- For each ring, we randomly select 100 samples in the circunference. For each sample, we add noise with the following equation:

$$X_{\text{noise}} = X_{\text{ring}} + \text{randn}(0, 1) \cdot \text{noise\_level} \tag{22}$$

  Where $X_{\text{ring}}$ is the set of points in the circunference, and noise_level is the noise level, and $\text{randn}(0, 1)$ is a random vector sampled from a normal distribution with mean 0 and variance 1.
- To add the background noise, we select N points in a rect centered at $(0, 0)$ with sides of length 1200, sampled from an uniform distribution.

For the sake of generality, we used consistent hyperparameters for the general experiments.

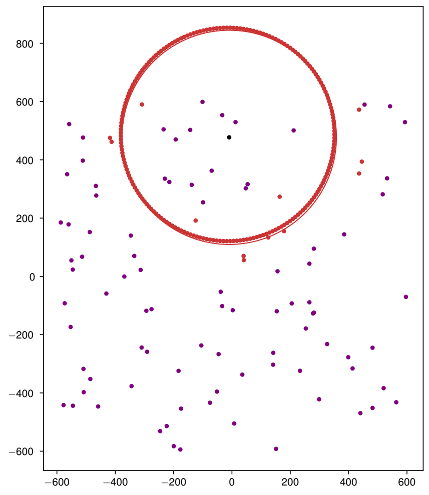

Fig. 9. Example of a dataset with a ring in a lot of noise, and background noise, and a good classification.

*3) Needle in the Haystack - Finding a ring in a lot of noise:* Another task we were interested about was the algorithm ability to, in a dataset with a lot of noise, find a ring. We used the same generation method as the general excentric test, but with 200 non-noisy samples for the ring and 100 and 200 background noise samples. We did not try more noise samples because then the ring would be almost irrecognizable. The results can be seen in the respective table.

*4) Is the noise detection effective?:* We conducted a test to see if the noise detection was effective. We used the same

| Number of rings | Ring noise | Background noise | Avg. Error | Avg. Runtime | Iterations | Experiments | Avg. Detected Noise |
|---|---|---|---|---|---|---|---|
| 2 | 0 | 0 | 2.46091 | 0.00128263 | 3.66667 | 3 | 0 |
| 2 | 10 | 0 | 8.34515 | 0.0012588 | 4 | 3 | 0 |
| 2 | 10 | 20 | 37.2104 | 0.652302 | 3338.67 | 3 | 53.3333 |
| 3 | 0 | 0 | 8.14361 | 0.998159 | 3336.33 | 3 | 0 |
| 3 | 10 | 0 | 8.26072 | 0.00172157 | 4.33333 | 3 | 0 |
| 3 | 10 | 10 | 12.296 | 0.029796 | 95.5 | 2 | 5.5 |
| 4 | 0 | 0 | 7.53772 | 0.0039321 | 7.66667 | 3 | 0 |
| 4 | 10 | 10 | 236.073 | 4.72157 | 10000 | 4 | 0 |

Fig. 7. Results of the general test with concentric rings.

| Avg. Error | Avg. Runtime | Iterations | Experiments | Avg. Detected Noise | Background noise |
|---|---|---|---|---|---|
| 4.24471 | 0.00246235 | 7.25 | 4 | 86 | 100 |
| 19.0933 | 0.0033355 | 10.3333 | 3 | 160 | 200 |

Fig. 8. Results of the needle in the haystack test.

| Number of rings | Ring noise | Background noise | Avg. Error | Avg. Runtime | Iterations | Experiments | Avg. Detected Noise |
|---|---|---|---|---|---|---|---|
| 3 | 8 | 200 | 38.912 | 3.83468 | 6560.3 | 20 | 248.15 |
| 3 | 8 | 200 | 68.1204 | 2.5356 | 4048.95 | 20 | 0 |

Fig. 10. Results of the background noise detection test.

generation method as the general excentric test, but with 200 samples for each ring, and 200 background noise samples, and a noise threshold of 70. The results can be seen in the respective table. As it can be seen, the algorithm detects, on average, more points than it should. However, the error rate is considerably lower than the one without noise removal, at the expense of a higher runtime.

*5) Experiments conclusions:* We can see that the algorithm is able to classify ring-shaped datasets with noise, and is fairly good on noisy data, but it degrades with higher noise levels and higher number of rings. Even if the average results are overal good, upon visual inspection, it can be seen that it either does terribly, or does very well. We hypothesize that this is due to the fact that once it has an erroneous classification, it is hard to recover from it, and it 'explodes'. Specifically, we can see that, when clustering concentric rings with background noise, it tends to 'join' rings together and treat the noise as another ring.

## V. Conclusion

In this paper, we have presented the Fuzzy K-Rings algorithm, in a setting where we had to classify ring-shaped datasets with noise. Based on previous literature, we reformulated the algorithm, and extended it with a noise removal step, to make it more robust. As our experiments tell, the algorithm is able to classify ring-shaped datasets on 'good' data, and is fairly good on noisy data, but is sensitive to hyperparameters and its performance degrades with higher noise levels and higher number of rings. Given the parallel nature of the algorithm, further research could be done to parallelize the algorithm and run it on GPUs, which could be easily done with GPU frameworks, such as PyTorch [8].

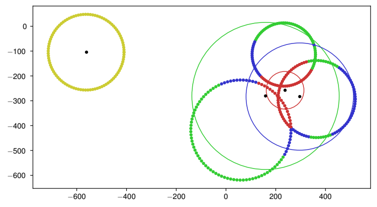

Fig. 11. Example of the algorithm 'exploding' in a bad classification. Instead of having errors in the classification, it just does something completely wrong.

## References

[1] Y. Man and I. Gath, "Detection and separation of ring-shaped clusters using fuzzy clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 8, pp. 855–861, 1994.

[2] R. N. Dave, "Generalized fuzzy c-shells clustering and detection of circular and elliptical boundaries," *Pattern Recognition*, vol. 25, no. 7, pp. 713–721, 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0031320392901345

[3] N. B. I. Pratiwi and D. R. S. Saputro, "Fuzzy c-shells clustering algorithm," *Journal of Physics: Conference Series*, vol. 1613, no. 1, p. 012006, aug 2020. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1613/1/012006

[4] R. Krishnapuram, O. Nasraoui, and H. Frigui, "The fuzzy c spherical shells algorithm: A new approach," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 663–671, 1992.

[5] J. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms*, 01 1981.

[6] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers & Geosciences*, vol. 10, no. 2,

pp. 191–203, 1984. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0098300484900207

[7] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.