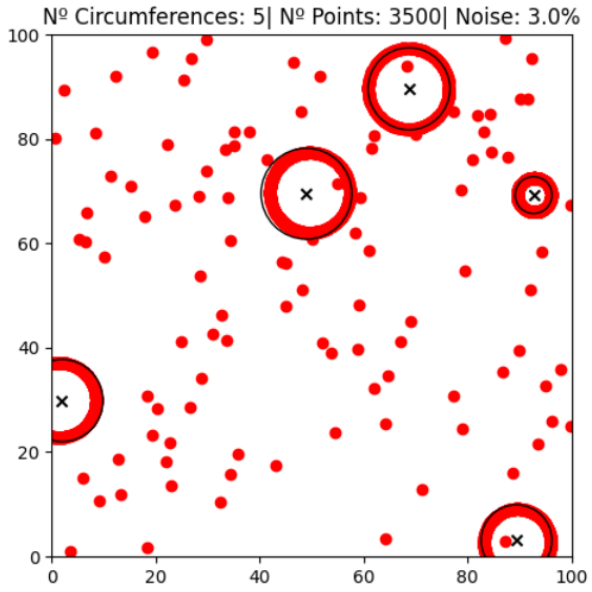
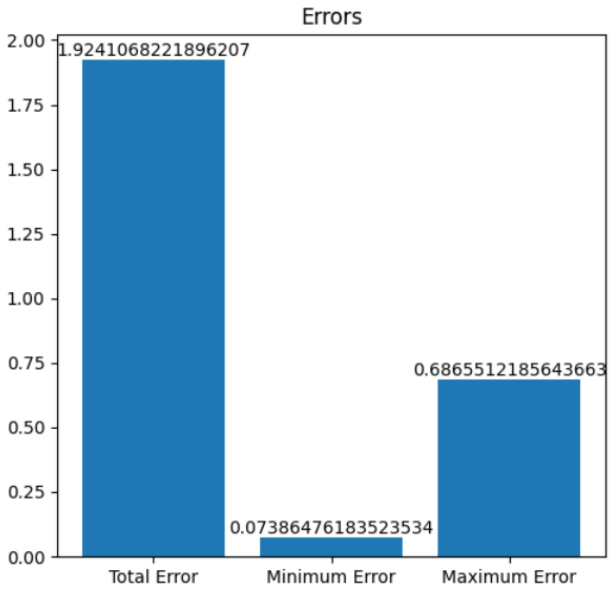
Artificial Intelligence

Clustering Noisy Rings

Github: https://github.com/Marchabar/clusteringNoisyRings





Index

1-Introduction

This project introduces a Python-based algorithm designed to search for circumferences, taking uncertainty into account. This report provides a comprehensive explanation of the algorithm, along with details of the conducted experiments and the resulting conclusions.

For additional information, please refer to the GitHub repository.

2-Contents

In this section, an explanation of the algorithm, as well as an analysis of the conducted experiments, will be presented. Regarding the source code, if you wish to try the algorithm yourself please check the README.md of the github repository.

a. Explanation of the algorithm

The idea behind this algorithm was partly inspired by the well-known Fuzzy C Means algorithm, which already considers memberships. However, I want to clarify that I developed the entire source code myself. To achieve this, I had to read several reports explaining how the algorithm works and performs. I also watched videos about different types of algorithms to familiarize myself with this approach about clustering algorithms.

You can find all the resources I used in Section 4, the Bibliography. Combining my knowledge from the Artificial Intelligence course with what I learned from my research, I was able to develop the following algorithm:

i. Initial data generation:

It first asks for the number of circumferences you want to draw, as well as the number of points that the plot will use to generate the aforementioned circumferences, and the noise you want to introduce to the algorithm. This is done by the GUI developed using the tkinter library, and there are upper limits for each of the inputs just for the statistics-purpose so I have clear which range I have to cover as much as varied experiments (more on this later). We have to take into account that it implements an automatic instance generator and that there is error validation for the cases in which the user provides inputs out of the ranges.

Once this is set, it first generates random centers in (-100,100) for the generation of the initial circumferences, and it plots random radiuses around them, with values in (3,8) so they are neither very big nor very small. And finally plots the points forming the expected circumferences , as well as the noise points stated before.

ii. Searching for circumferences:

Taking the above inputs, the algorithm first generates random memberships of each of the points for each of the circumferences, and this is normalized so it sums 1.

Then the centroids are computed using the formula of the Fuzzy C Means, and this is done for each of the circumferences.

It continues by computing the actual radiuses from each point to the centroids computed before, finally computing the new memberships with respect to the new centroids using the Fuzzy C means formula.

All this is done once and again until the maximum number of iterations (I found 300 as a proper value, fuzzines value m is 1.1, and the convergence error is 1e-5) is reached or until the difference between the memberships converges.

As a result, we have the centroids (centers of the identified circumferences) and the radiuses. This result is shown in the final plot.

iii. Error computing:

To enhance comprehension and refine the algorithm, I calculate the errors between each real circumference and the identified circumferences. This allows for the generation of statistical insights and a deeper understanding of the algorithm's performance.

The total error represents the sum of each of the errors between each centroid and its closer real center. Even though this is not so real, it helps us get an idea of how the algorithm has performed for each of the experiments.

The minimum error represents the smallest error between the centroids and the real centers for this experiment.

The maximum error represents the biggest error between the centroids and the real centers for this experiment.

These errors are later shown in the final plot, as well as marked in the final statistics table.

iv. Saving the results:

The outcome of the final plot is presented in the form of two subplots. The left subplot depicts the final rendering of the actual circumferences, along with the initial noise and the identified circumferences, visually differentiated by various colors. On the right-hand side, a bar graph is used to display the errors, enabling easy observation of their respective proportions. Each plot can be found in the "plotFigures" folder.

Furthermore, the "csvFiles" folder is included, which stores the initial point coordinates for each new experiment that is executed.

Lastly, there exists a file named "table.csv" that serves as a repository for the primary information pertaining to each execution and its corresponding errors. This file serves as the statistics table and encompasses a total of 50 experiments, aimed at encompassing a diverse range of inputs.

This entire implementation is coded in Python, with comprehensive commenting throughout the source code to facilitate ease of understanding.

b. Overview of the experiments

To facilitate the analysis, the algorithm considers a total of 50 experiments contained within the "table.csv" file. This file serves as a valuable resource for gathering data and conducting comprehensive evaluations. By including these 50 experiments, the algorithm ensures a robust and representative dataset for the analysis process. Each experiment likely provides unique insights and contributes to a more comprehensive understanding of the subject at hand.

Considering that it takes 3 inputs: number of circumferences, number of points and noise percentage, the experiments are executed from 1 to 10 circumferences, from 50 to 6000 points and from 1% to 10% of noise.

Just below, the table with all the experiments runt can be found:

numCircumferences	numPoints	noisePercentage	totalError	minError	maxError
1	50	5.0	2.7194632330143618	2.7194632330143618	2.7194632330143618
1	50	5.0	2.660299065183894	2.660299065183894	2.660299065183894
2	200	5.0	3.3533337552698024	0.08044429208674987	3.2728894631830525
2	200	5.0	4.4053563116979975	1.5921630756264122	2.813193236071585
3	500	5.0	5.460667986732499	0.21292138716614067	2.8654010704974024
3	500	5.0	3.030083044426092	0.34265869680967626	1.7725124065213234
4	1500	5.0	2.842703749884518	0.03471031226531986	1.9426009838357559
4	1500	5.0	14.737492853877823	0.06618982398627268	9.238795506594695
5	2500	3.0	2.5304702446539564	0.16660127918563816	0.7374259967691841
5	2500	3.0	1.4398366214928044	0.06588169590522441	0.4844109636573792
6	3000	4.0	45.113705250783795	0.22879870063077815	33.54389003525778
6	3000	4.0	80.34020070926186	0.21182421108093455	51.52016989840235
7	4000	5.0	40.43218595688445	0.05710191987743655	30.189953399405056
7	4000	5.0	8.94238934841747	0.15837324433336544	4.7471174193331755
8	5000	5.0	20.329430658359712	0.23015679366201136	6.960683403945339
8	5000	5.0	24.754660437040236	0.1592094720523251	11.65118056572165
9	5500	2.0	16.156429614453444	0.016260898569532424	4.918151980281893
9	5500	2.0	14.681201497595026	0.08223011228562467	6.987280177343508
10	6000	1.0	20.579236418578553	0.016936395174951	9.343948593071245
10	6000	1.0	11.208441053919389	0.057570379157278584	4.947622565382747
1	100	10.0	3.788393248289458	3.788393248289458	3.788393248289458
1	500	6.0	1.2360725647901505	1.2360725647901505	1.2360725647901505
1	4000	30.0	14.03634469054771	14.03634469054771	14.03634469054771
2	700	8.0	4.455251188192406	2.1140315051254306	2.341219683066975
2	2000	5.0	2.629135125699637	0.8044196926620829	1.8247154330375543
2	4000	10.0	1.7277069203219373	0.705694576075984	1.0220123442459532

3	450	3.0	1.599203939045131	0.2029553269820683	1.1905295369520854
3	5000	4.0	3.4106446532537404	0.2690737407986105	2.6366406177689763
4	800	2.0	1.1880336866714725	0.05422381722650653	0.7162784290057975
4	5000	4.0	3.3220467730992205	0.23120882848116053	1.4247589398882166
4	6000	7.0	5.471173230824874	1.2169040162094393	1.4909999300589052
5	1000	1.0	11.551087949014534	0.1689195955071412	4.316637162975386
5	3500	3.0	1.9241068221896207	0.07386476183523534	0.6865512185643663
5	5800	3.0	1.4943749692062245	0.0743086375024633	0.6703004076594402
6	300	2.0	32.87372633407049	1.9586631705970035	16.166079817415277
6	3000	2.0	4.586178920334035	0.16393190530575663	2.1129764634581805
6	6000	7.0	3.6007886302456535	0.23635500245601856	0.9283329398751329
7	4250	3.0	62.59711083100503	0.08644842967308984	52.55740290397381
7	5000	3.0	19.51654423316893	0.3732842980380814	5.594042665156422
7	6000	3.0	17.794969987430626	0.21249275334129158	5.9488214140095454
8	5000	2.0	1.7203675786522803	0.04926671304569622	0.44158340609074026
8	5500	8.0	33.25643061039134	0.077722824392352	28.240419385995533
8	6000	2.0	21.96681099019197	0.08484582019798734	8.60365307365076
9	5000	3.0	31.611517887644723	0.0316006004151222	7.267622490528573
9	5500	3.0	35.15756362564464	0.05828448367146023	21.230587191379843
9	5999	2.0	31.039745015895072	0.024377752851351287	23.998175626460206
10	3500	2.0	23.49386896326356	0.03988616349079326	4.326976346228012
10	5500	2.0	16.97979939011735	0.1060227250357583	5.1440816113940535
10	6000	10.0	18.014853267167688	0.28765722262153276	6.696772686641133

As observed, initially there are two records per same number of inputs, allowing then multiple executions for each input.

Subsequently, three executions are performed per number of circumferences, involving variations in the number of points and the percentage of noise, thereby yielding diverse results.

The medium/big size execution is the last entry of the table, where all the upper limits for each of the inputs are set, in this case 10 circumferences, 6000 points and 10% of noise. This plot result can be checked as the last entry in the "plotFigures" folder. Take into

consideration that the “plotFigures” folder contains all the plots of all the 50 experiments runt.

3-Conclusion

In general, the algorithm demonstrates strong performance when there are no overlaps among the generated circumferences. This means that it yields the best outcomes when dealing with a small number of circumferences, as the likelihood of overlapping is lower. Additionally, the number of points plays a significant role, and for a small number of points, the algorithm's performance is not as effective. This is because the algorithm employs a weighted average approach.

However, when there is overlap or very close circumferences, the algorithm mistakenly perceives the overlapping circumferences as a single larger circumference. Consequently, it draws the remaining circumference in areas where more noise points are present, assuming that it must draw as many identified circumferences as there were initial circumferences. As a result, the algorithm often fails to correctly handle overlapping circumferences, but it performs pretty well when overlaps are absent.

4-Bibliography

Day 71 - Fuzzy C-Means Clustering Implementation

K-Means Clustering Explanation and Visualization

Training Clustering Algorithms

Machine Learning | Fuzzy C Means

https://www.analytixlabs.co.in/blog/types-of-clustering-algorithms/

https://www.mathworks.com/help/fuzzy/fuzzy-c-means-clustering.html

https://sites.google.com/site/dataclusteringalgorithms/fuzzy-c-means-clustering-algorithm

https://www.mygreatlearning.com/blog/dbscan-algorithm/#:~:text=DBSCAN%20is%20a%20clustering%20algorithm,separated%20by%20low%2Ddensity%20regions.

https://www.geeksforgeeks.org/ml-fuzzy-clustering/

https://www.geeksforgeeks.org/ml-k-means-algorithm/