Michael Fulton
CS445 Winter 2022
Program 3

K Means and Fuzzy C Means classifier

In this program I explored a pair of clustering unsupervised learning algorithms. The dataset is a matrix of [x,y] data points that represent simple clusters. Apparently the data was created from 3 Gaussian mounds which have considerable overlap. As a result, neither of the algorithms used will be able to extract the meaning of the clustering. However, some of the results do result in some pretty good looking results. There were a number of parameters as well which were tweaked to see what kind of results I could get, and I created some visualizations from the data too.

How to Run

To run the program you will need a python interpreter, as it is written in python. I used the numpy library, as well as matplotlib for the data visualizations, so you will need those in your environment as well.

To execute the program and get the results, you simply need to run the python file:

```
$python3 k_means.py
```

After the program has finished executing, the results are a series of PNG plots located in the km, and cm folders for K-Means, and Fuzzy C Means results. The resulting PNG files indicate the number of centroids, and which epoch the picture is for.

By default the program runs with a fuzzifier of 2, and runs over 10 epochs. Additionally its setup to run through these with 2, 3, 5, and 7 centroids. Changing the parameters is fairly straight forward and can be done by editing a couple of well labeled values in the main routine.

Design Considerations

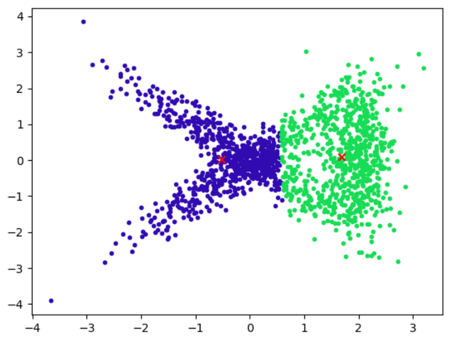
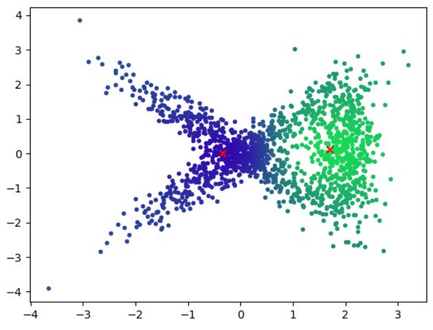
There were a number of things I had to decide for when setting up this program. First of all, the actual results are basically just the centroids. On their own, a couple of x, y coordinates don't have much meaning. So I decided to use matplotlib to create colorized scatter plots of the resulting classifications. After I collected the data, I turned these scatter plots into animated GIFs so we can see the overall results. Some of these GIFs are included along side this report. Random colors were used, so the contrast between classes is sometimes less than ideal. If I were to spend more time on this, working out a better scheme for the colors would be one of the things that I would do.

For the Fuzzy C Means the visualization is a bit different, as each point has partial membership to each class. To represent this, I calculated the average color based upon the membership rate for each classification. This resulted in a rather slow algorithm to run, as the matplotlib call to add a particular point was kind of slow when adding one at a time. But the overall run time of the program still isn't that long.

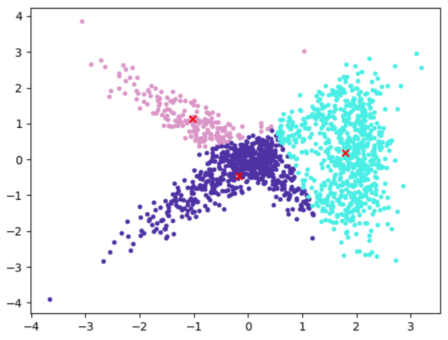
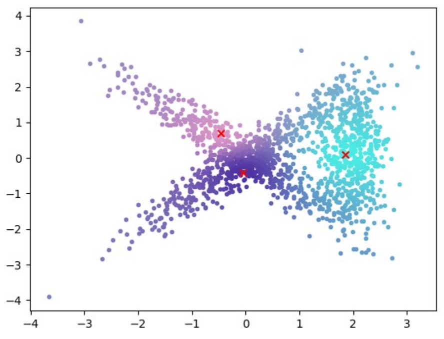
The final main consideration was what to do with the hyper parameters. I found that the number of epochs would usually converge pretty nicely in under 10, so I set the limit to that. It easily could have been bumped up to 20, but for most runs of the program, it wouldn't have made much of a difference. The other one to consider was the number of centroids to test. The data is from 3 Gaussian mounds, so fewer than 3 seemed low, but I started with 2 none the less. I tried up through 10, and the data seemed to turn into a mess around 7, so that was the highest value I kept in. The 'm' fuzzifier, was probably the most difficult one to tweak right. Until I had the average color visualization working, it was hard to tell what the fuzzifier really was doing. Values of more than 3 seemed to result in very smeared out classifications. Values close to 1 resulted in very sharp lines. I liked the results I got with about 2, but I think a value of 1.5 would have looked pretty good too. So I stayed with an fuzzifier value of 2.

Results

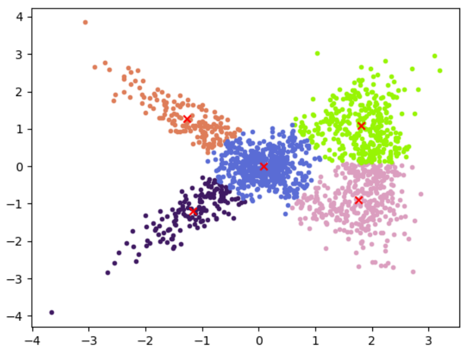
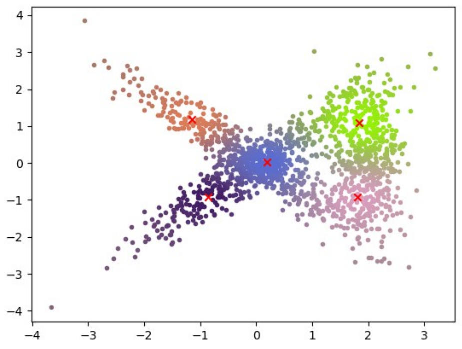
The results for running this are best explored by the plots it created. So I'm going to pick a few of my favorite results, and talk about them. Do be sure to check out the animated GIFs provided along with this PDF document for more results. For both the fuzzy, and the regular K means runs I had them start with identical starting points, and same color schemes so I could best compare the results. This does result in fairly similar results between the two most of the time, but it also is the best direct comparison.

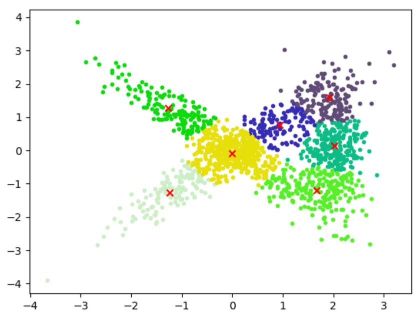
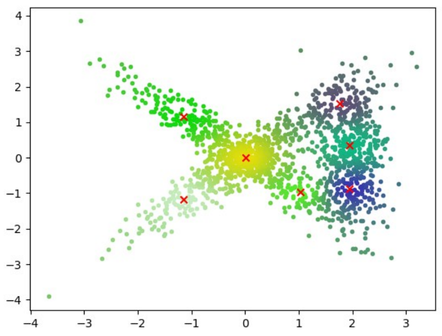
The above images are results with two centroids. The data does split fairly well between these two spots, but there is a lot of nuance in the data that is missing. I think this best suggests that there are more than 2 classes here to deal with.

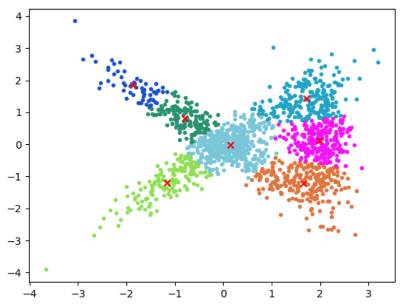
Here with 3 centroids we are at least capturing some of the features in the data, that there is a split on the left side, and that the right side mostly belongs to a single class. However, knowing that this data comes from 3 mounds with a lot of overlap, we're still missing some of the information in here.

Here with 5 points I think we've finally captured more of the data features. The two arms are their own class. The overlap in the center is its own class, then the overlapping 3rd mound that intersects with the two arms on the right are their own class. I think I might have benefited from running this with 6 classes, but I went straight to 7 for the final run of this. Additionally, here we can see the usefulness of the fuzzy classifier, as its able to show where this is more and less overlap between these classes pretty nicely.

Finally, here with 7 centroids we actually see fairly different classification results with the Fuzzy and K means. We're actually capturing a bit of the differentiation in the overlap on the right side of the graph, but if anything we might have one too many centroids.



Also, this was the number of centroids where I got the most variation from run to run. Some clearly were inferior to others. Here's one example where an arm on the left ends up with two different classes.

Conclusion

I think the results that were produced here were very good. As was stated in the lectures, these algorithms do suffer some from falling into local minimum, and require a lot of tweaking of the hyper parameters. Doing this can produce some results that are pretty darn good, but clearly require a little manual verification. It took a lot of tweaking to come to the conclusion that 5 or 6 centroids work best with the data, that the number of epochs should be at least 10, and that the fuzzifier should should be between 1.5 and 2.5. And even after that, these results are somewhat subjective. Anyway, I'm interested in exploring how these algorithms are useful tackling other problems, and data sets.