# Programming Assignment 3

Shannon Keola

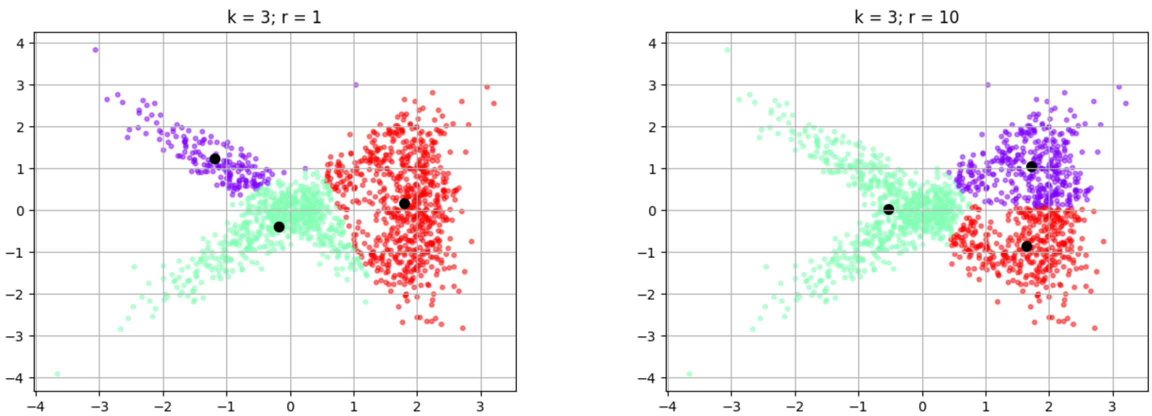
November 29, 2019

# Experiment 1: K-Means

In this experiment, we will be testing the K-Means classification algorithm using Python's `numpy` and `matplotlib` libraries. We will be using a 2d data set taken from 3 Gaussians, 500 points each, with considerable overlap. Each result is tweaked by two hyperparameters, $k$ and $r$. $k$ is the number of clusters (k-means) and $r$ is the number of iterations we run before taking the solution with the minimum within-class sum-of-squares error. For each value of $k$, I chose to use two values of $r$ ($r = 1, 10$) because there was little to no variation when I tested with other values, namely 5 and 50.

**Results:**

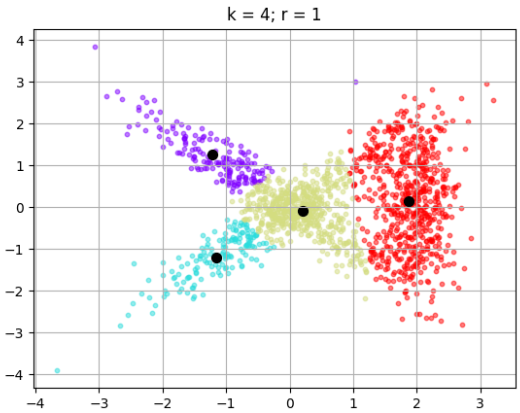
For each value of $k$, there are two scatter plots with each cluster shown by a different color and a big, black dot in the middle to represent the cluster's mean. While running for various values of $r$, the different initializations usual led to different tiers of WCSS error. For example, when using $r = 10$, the algorithm usually found the lowest WCSS during one of the iterations, but for $r = 1$ it was random. For the sake of comparison, I ran the $r = 1$ a couple times to get a higher WCSS than the $r = 10$ iteration. The graphs of my results are below:



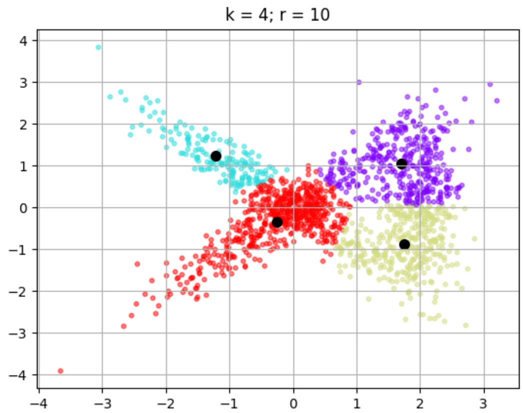

(a) WCSS = 1389.60

(b) WCSS = 1297.31

Figure 1: k=3

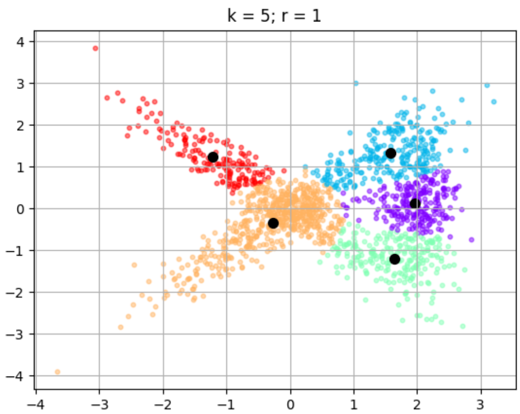



(a) WCSS = 1223.07

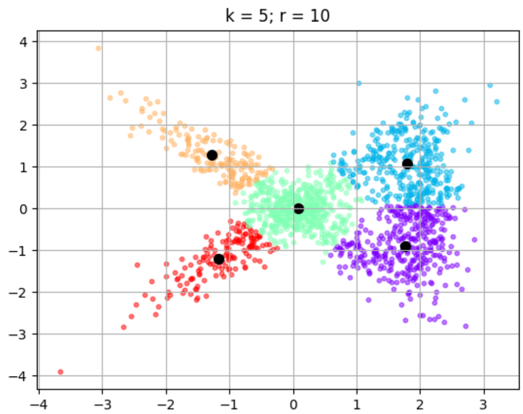



(b) WCSS = 1110.98

Figure 2: k=4

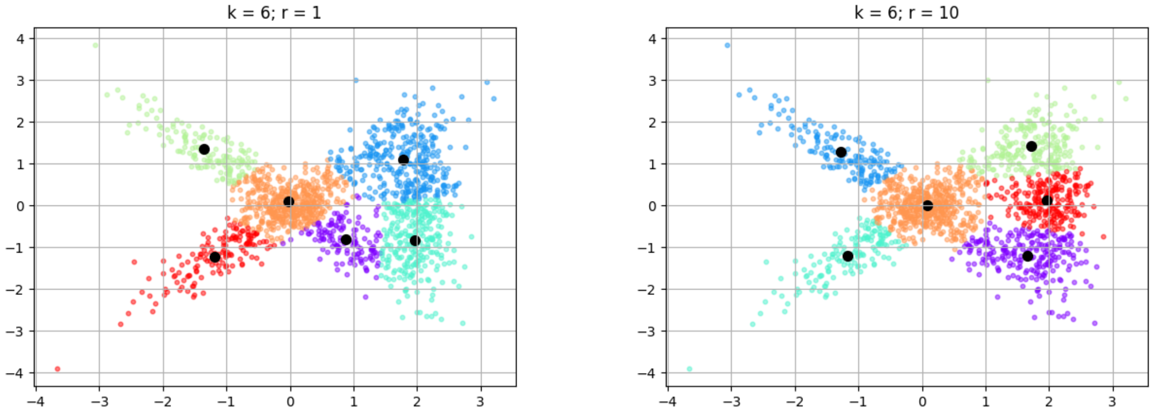



(a) WCSS = 1010.53




(b) WCSS = 947.74

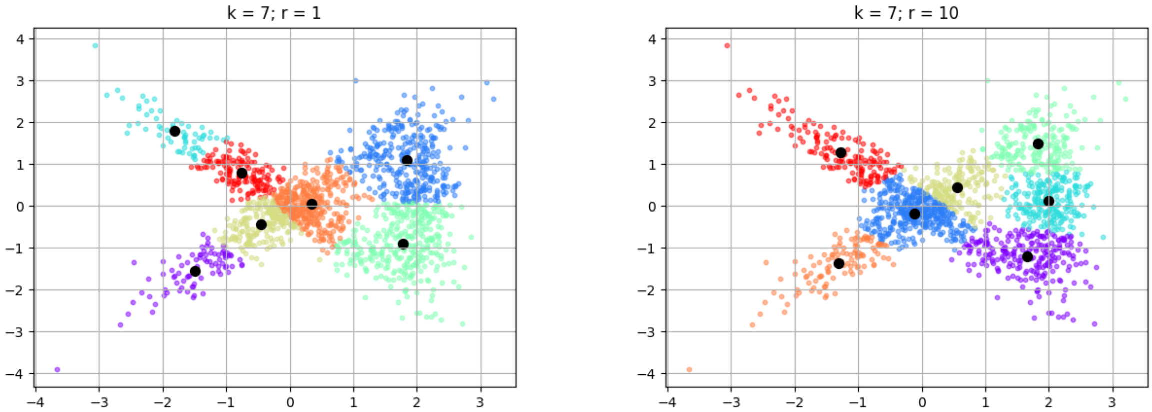
Figure 3: k=5




(a) WCSS = 895.18

(b) WCSS = 846.17

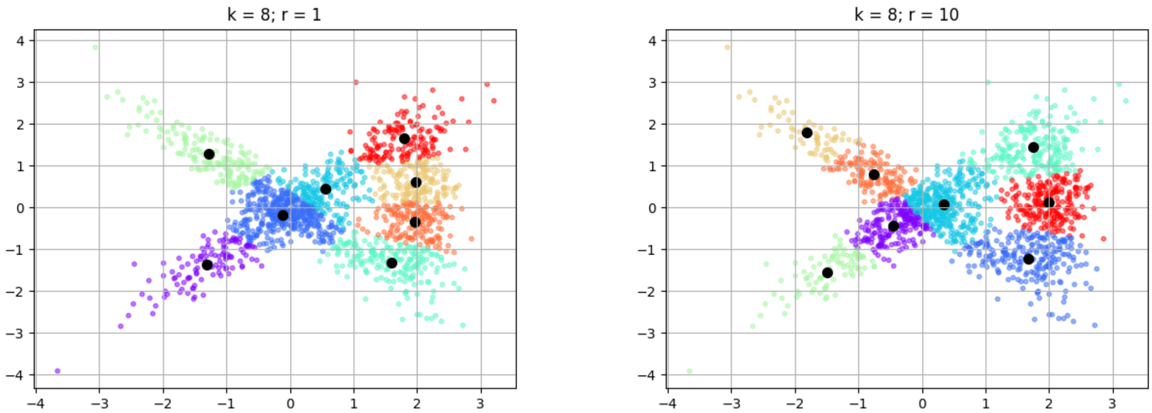
Figure 4: k=6




(a) WCSS = 848.67

(b) WCSS = 793.99

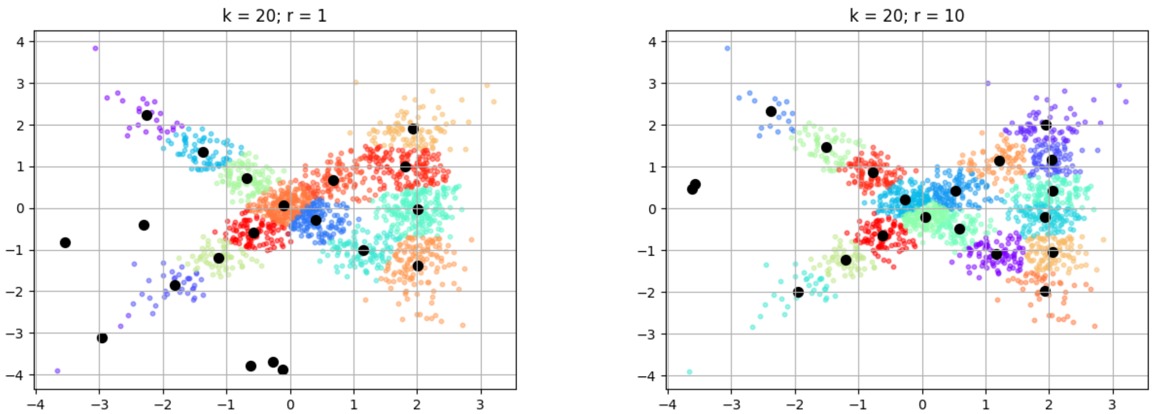
Figure 5: k=7




(a) WCSS = 760.14

(b) WCSS = 747.42

Figure 6: k=8




(a) WCSS = 559.29

(b) WCSS = 484.64

Figure 7: k=20

**Analysis:**
From the graphs above, we can see that the WCSS error decreases as we increase the number of clusters ($k$) and as we increase $r$. The decrease in WCSS as we increase $k$ is automatic because as we increase the number of clusters, the size or spread of each cluster decreases. The decrease in WCSS from $r$ is not always true as we could hit the optimal solution on the first run and each following run will not decrease WCSS. However, on average, increasing $r$ decreases WCSS error. I went a bit overboard with the number

of iterations because the graphs were coming out nice and I wanted to stress test my K-Means algorithm as well as my graphing algorithm. In Figure 7, we can see an example with $k = 20$, where some centers have no points in their cluster. This is because if a randomly initialized cluster has no points that are closest to it, it never shifts toward the data and gets 'left behind'.

## Experiment 2: Fuzzy C-Means

In this experiment we will be testing the fuzzy c-means classification algorithm. The dataset is the same as the set from the k-means experiment above. For this experiment, I chose to break it down to identifying the effects of varying two parameters, $c$, the number of clusters/centroids, and $m$ or the 'fuzzifier'. I also had a parameter $\epsilon$ that represented the sensitivity of the stopping condition ($\Delta centroids$), which I fixed to 0.0005 as well as $r$, the number of randomly initialized iterations to 3. This helps to prevent extremely long runtimes by reducing the number of iterations spent calculating miniscule deltas as well as oscillations.

**Varying $c$:**
For this portion of the experiment, I fixed $r = 3$ and $m = 2.0$ for the entirety. In my graphs, each point is colored by using a weighted sum of squares of each centroid's color. This results in points with high weights for one centroid and low weights for the others to have a color that is nearly identical to the centroid. What we see then is that points very close to a centroid have a 'true' color, while the fringe points resemble a more muddled, black color. I've also included a convex hull for each centroid, representing the points for each cluster that have a weight greater than 0.5. The results are below:

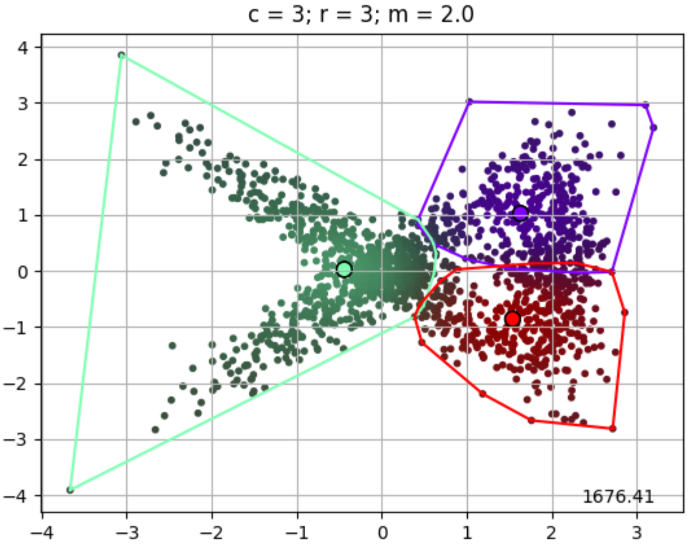



Figure 8: $c = 3$

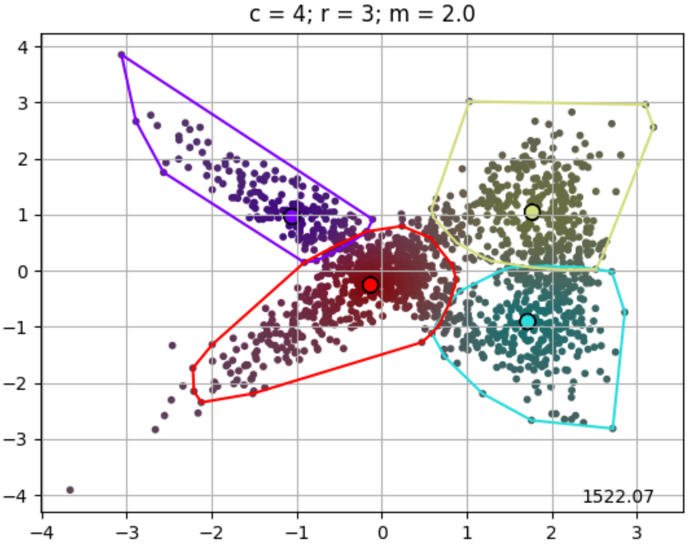



Figure 9: $c = 4$

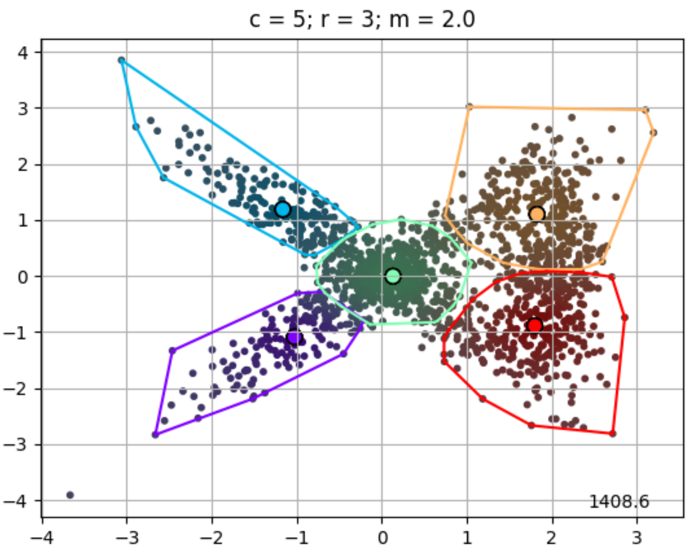



Figure 10: $c = 5$

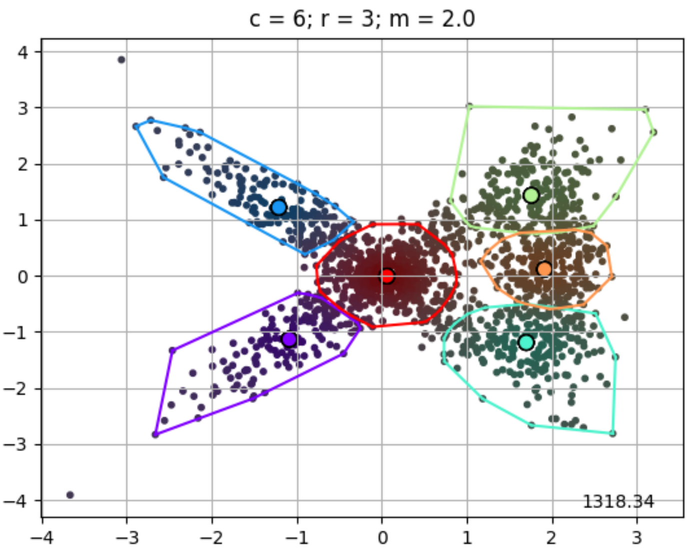



Figure 11: $c = 6$

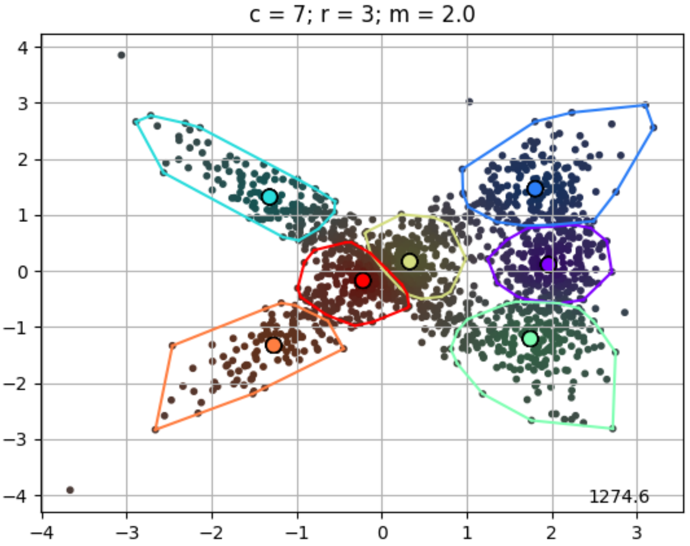



Figure 12: $c = 7$

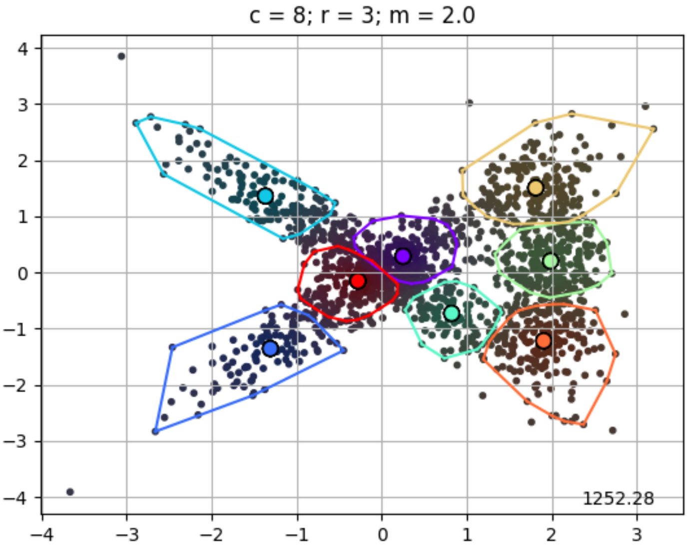



Figure 13: $c = 8$

We see that as we increase the number of clusters, we get a decrease in weighted WCSS error, pictured in the bottom left of each graph. We also see that many points, including ones very close to a centroid, begin to lose their color intensity, as each point has a 'probability' of being in more and more clusters. We also start to see that the weighted WCSS error decreases more slowly as we increase $c$, similar to the k-means experiment.

**Varying $m$:**
For this part of the experiment, I will be varying $m$ for 3 different values ($m = 1.1, 2, 3, 5$). I will be fixing $c = 4$ and $r = 3$ to see the effects of $m$ on our weights and centroids. These graphs also include the convex hull and colored weights for visual effect.

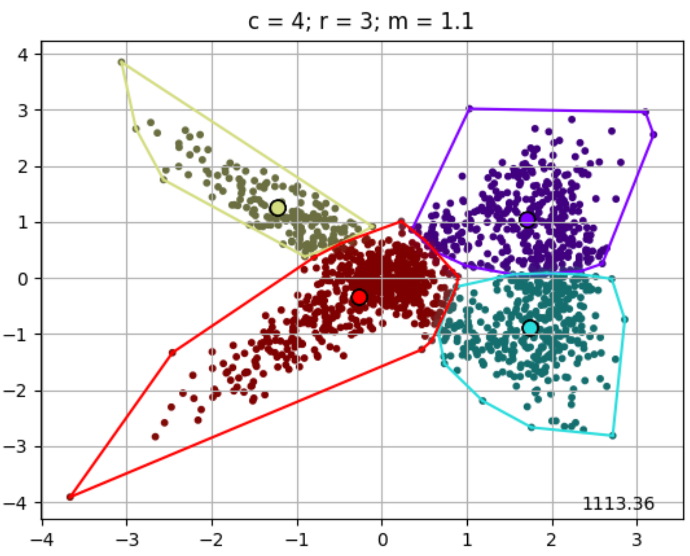



Figure 14: $m = 1.1$

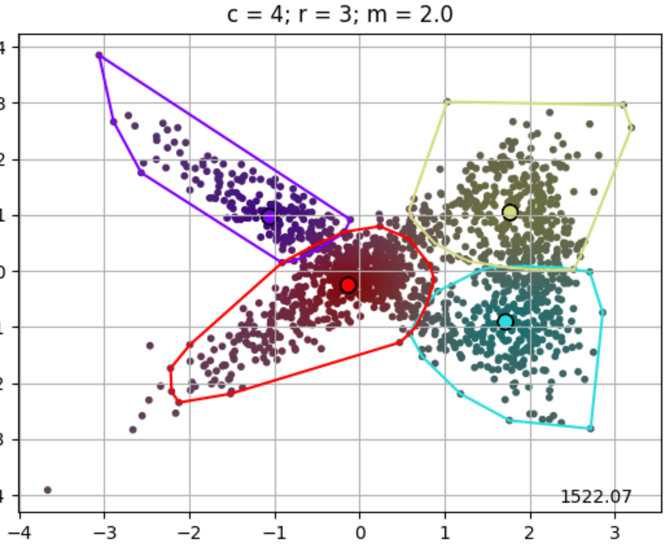



Figure 15: $m = 2$

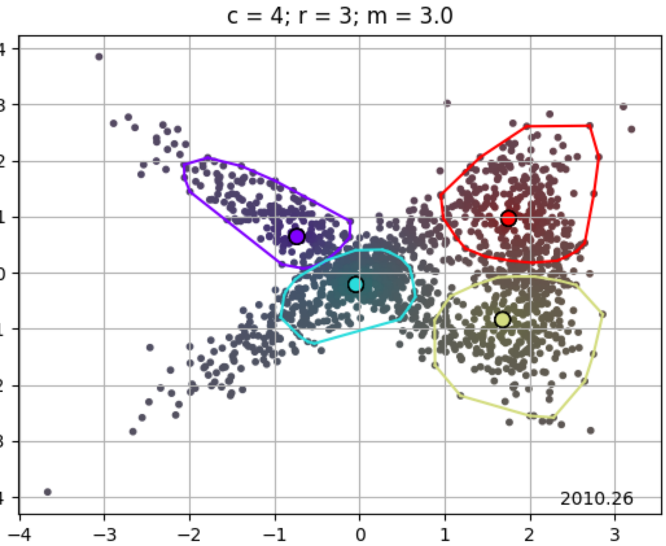



Figure 16: $m = 3$

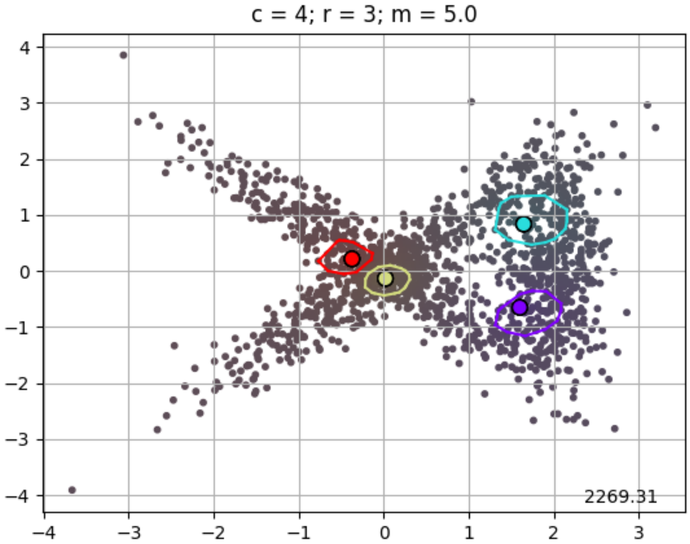



Figure 17: $m = 5$

We can see in the graphs above that as we increase $m$, the 'muddled' or uncertain points grow in number as the classifier becomes more and more fuzzy. When $m = 1.1$, the graph almost resembles the k-means classifier, while $m = 5$ gives us a classifier that can only classify points very close to a centroid. We can also see the convex hull ($w > 0.5$) shrinks as we increase $m$ as well.

**Code:**
I implemented these algorithms in Python using the `numpy` library. The k-means algorithm is driven by `kmeans.py` using the methods provided in `kmlib.py`. The fuzzy c-means algorithm is driven by `fcm.py` using the methods provided by `fcmlib.py`. To run k-means, use `python3 kmeans.py k n`, where k is the number of clusters and n is the number of randomly initialized iterations. To run fuzzy c-means, use `python3 fcm.py c r m e`, where c is the number of clusters, r is the number of randomly initialized iterations, m is the 'fuzzifier' parameter, and e is the sensitivity of the stopping condition.