Programming assignment 2: KMeans and EM

Part I: Implementation

(1) K-means

The basic idea of K-means algorithm is to repeatedly calculate the clusters' centroids and refining the values until the centroids stay at their positions and do not change much. The approach to implement this algorithm has three main parts corresponding to the three steps.

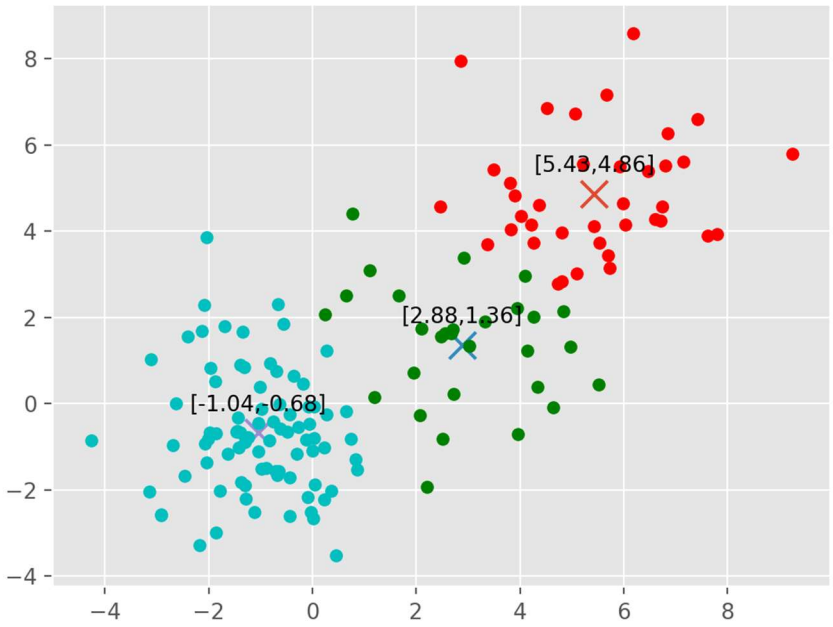
First, randomly choose K centroids. In this assignment K=3. In our program, they are saved in a list called "centroids". Second, assign each data point to its nearest centroid. To calculate the distance, we use Euclidean Distance to calculate the straight line distance from point to centroid. The equation is as following:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \; (where\ p = (p1, p2, \ldots, pn);\ q = (q1, q2, \ldots, qn).$$

For each point, we calculate its distance from all of the three centroids and pick the nearest one. The clusters of data points are packed in three lists and saved in a dictionary called "classes" with centroid's index as each cluster's key. Third, for each cluster, get a new centroid by using the "np.average" function which results in a weighted mean when giving a parameter. In this scenario, no parameter is needed. Then, repeat step 2 and 3 until the new centroids do not change much with the previous ones. Two breakpoints are set for the loop - tolerance and max_iterations. Tolerance is the difference between previous centroid and new centroid. In the program, it is set to 0.0001, which means if the distance between previous centroid and new centroid is smaller than 0.0001, the program considers they are close enough and the loop will terminate. Max_iteration is set to 500 according to the number of data points in this assignment in case for infinite loops.

After the first couple of runs, we found that there were always some points that were assigned to their clusters unreasonable. They seem closer to another cluster in distances. One possible reason is the initial choice of centroids. It makes the result vary. So, one of the optimizations is to choose different initial centroids several times and pick the best among the choices. This will lead our results closer to the optimal, and the more we try, the closer it would be. As a result, we introduce a dictionary to save previous results and cover it if the new try has better performance.

The final result is as following:



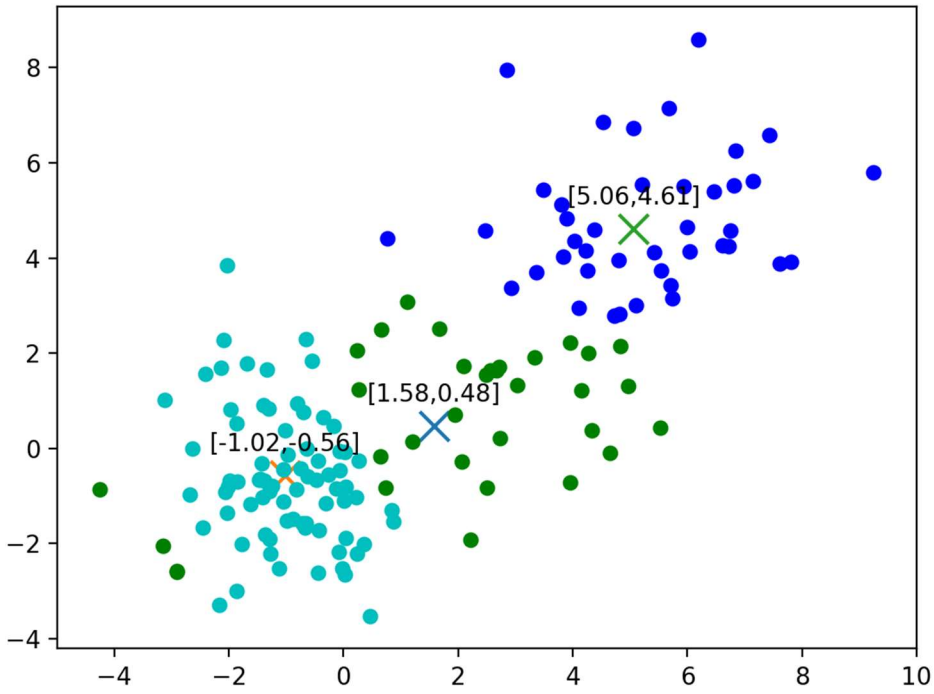
The 3 centroids are:

[5.43 4.86], [2.88 1.36], [-1.04 -0.68]

(2) Gaussian Mixture Models(GMM)

The Gaussian mixture modelling used Expectation-Maximization Algorithm to cluster the datasets into different clusters. It has rigorous functions to evaluate in each step. The algorithm keeps iterating “E-Step” and “M-Step” until the results converge.

The most difficult problem we faced is determining what is a “good start” for the algorithm. Different mean points in “E-Step” can affect the result due to the difference in calculation while clustering the points in the very first iteration. At first, we randomly selected points from datasets. However, the resulting cluster is unstable. Here are some samples from the code.



Centroids:

[1.57733127 0.4768599],

[-1.01980458 -0.56367116],

[5.06268222 4.61078049]

Amplitudes:

0.2918376567748539,

0.4417268133640977,

0.2664355298610484

Covariance Matrices:

[[6.79742391 3.20374914]

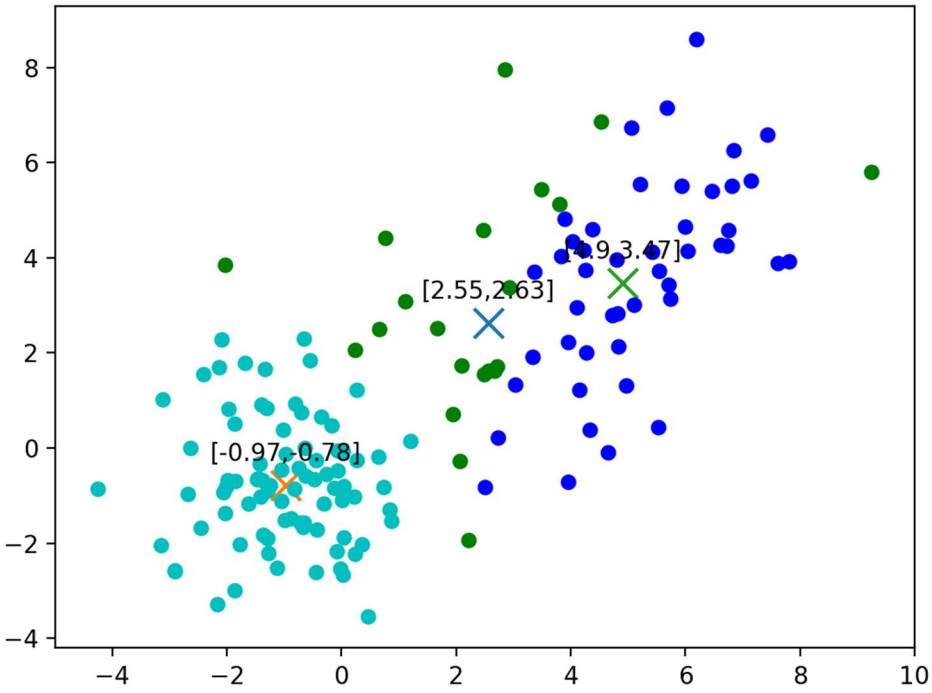
[3.20374914 3.5562077]],

[[0.8815942 -0.27285793]

[-0.27285793 2.06943535]],

[[3.0209779 0.78978479]

[0.78978479 2.59619792]]



Centroids:

[2.55482627 2.62596362],

[-0.97018192 -0.78001108],

[4.90441914 3.47267415]

Amplitudes:

0.24905807962566906,

0.5039501353540229,

0.24699178502030814

Covariance Matrices:

[[8.05528946 4.06415769]

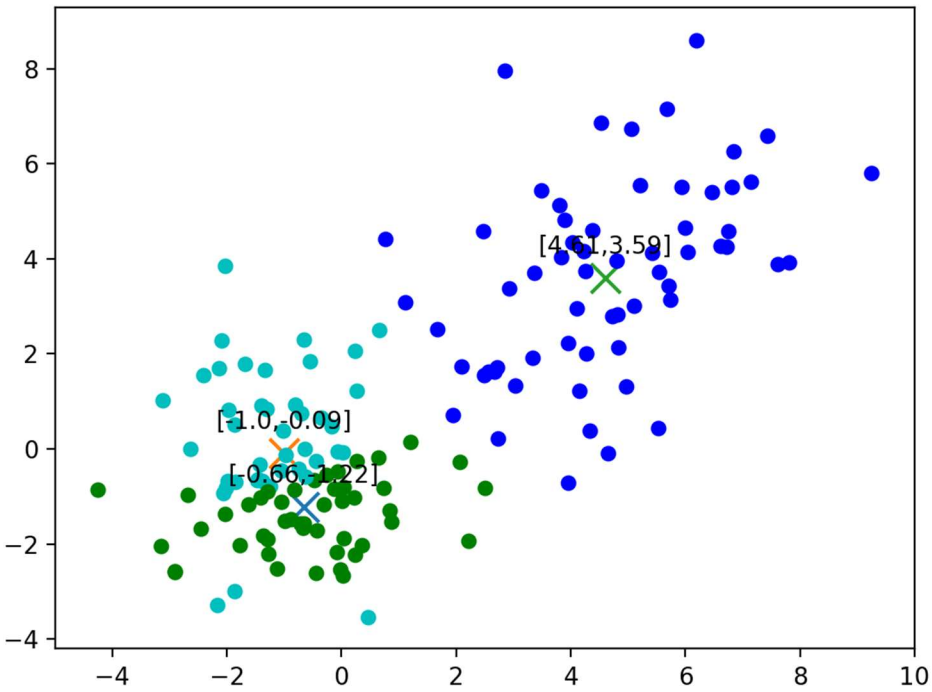
[4.06415769 6.34146695]],

[[1.08664887 -0.09381532]

[-0.09381532 1.63996782]],

[[2.41984119 1.99948913]

[1.99948913 4.47844631]]



Centroids:

[-0.65746335 -1.22468524],

[-1.00207196 -0.08940646],

[4.60529737 3.58937635]

Amplitudes:

0.2503169416509398,

0.3440492383444481,

0.40563382000461246

Covariance Matrices:

[[1.80428391 0.30337275]

[0.30337275 0.79721374]],

[[1.21126738 0.00708568]

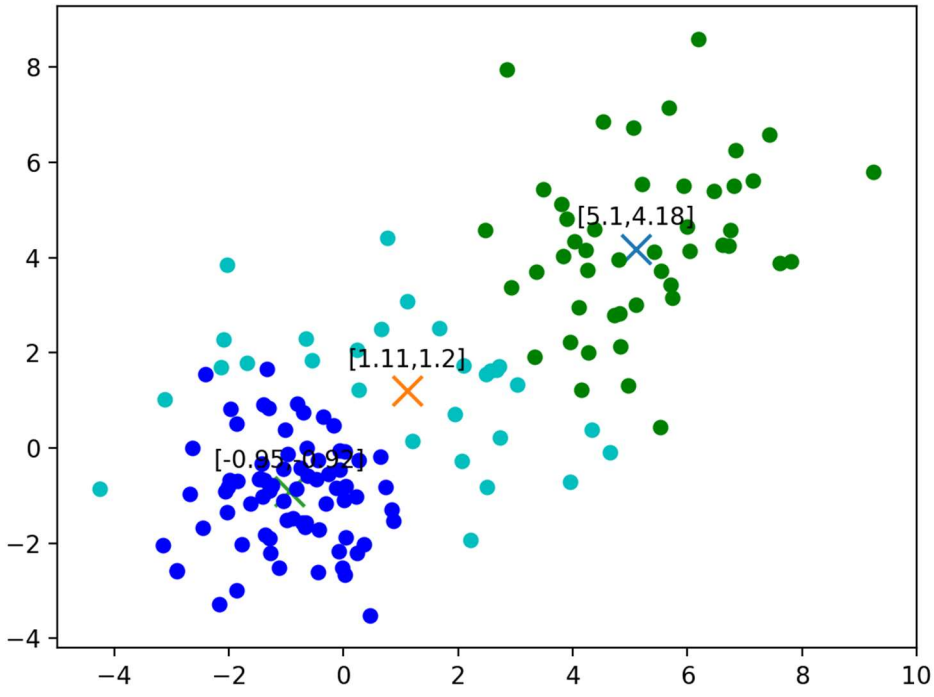
[0.00708568 2.64961287]],

[[3.12459004 1.72999967]

[1.72999967 4.43564237]]

As the graphs shown, the three clusters can vary a lot among different runs since the “start centroids” are chosen randomly.

Then we came up with the solution of using K-means to initialize the “start centroids” for GMM. The reason for doing this is by the fact that centroids resulting from K-means have converged. It is more stable than randomly choosing the start centroid. We did a few runs and the result proves our assumption. Most results fall into the graph shown below.



Centroids:

[5.10317991 4.17895959],

[1.11106709 1.1962312],

[-0.94727698 -0.91954266]

Amplitudes:

0.29635867403197996,

0.24918071794408808,

0.4544606080239319

Covariance Matrices:

[[2.51982284 1.15809753]

[1.15809753 3.72251765]],

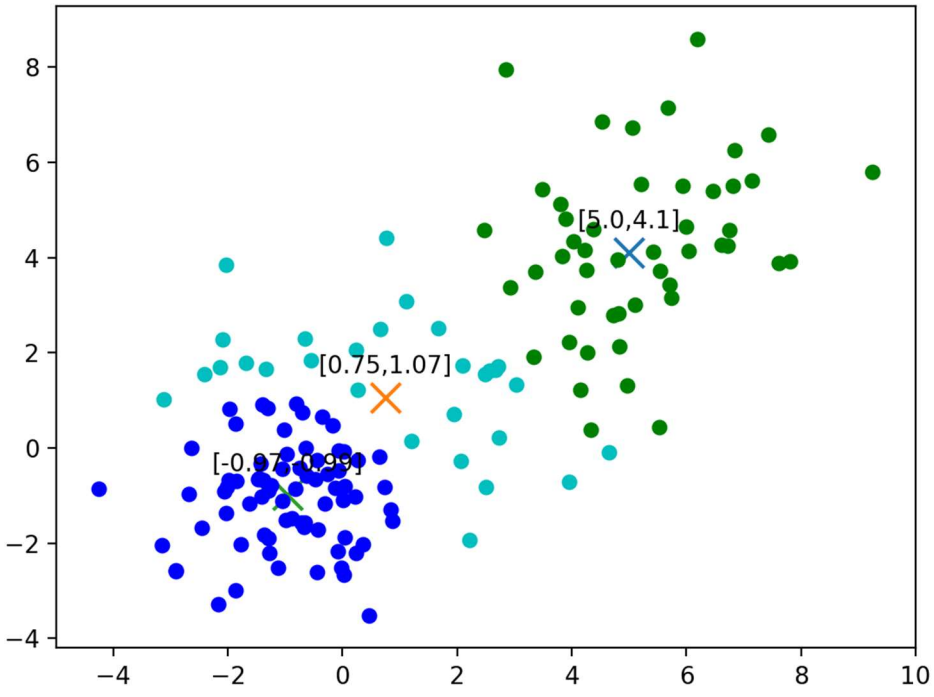
[[6.24887693 1.41331969]

[1.41331969 3.44277144]],

[[1.00185406 -0.05476076]

[-0.05476076 1.40207178]]

We do notice that there are noise points appearing on the left side of the graph, which is possibly resulted by the limit of max number of iteration. After increasing the maximum number of iterations from 10 to 30, the noise problem improved.



Centroids:

[4.99880818 4.10039532],

[0.75129312 1.06604865],

[-0.96634658 -0.99381011]

Amplitudes:

0.32817497980770466,

0.21393316298577797,

0.4578918572065174

Covariance Matrices:

[[2.63434655 1.17474786]

[1.17474786 3.63319793]],

[[5.08750656 0.0063713]

[0.0063713 2.39641696]],

[[1.07179691 -0.01510904]

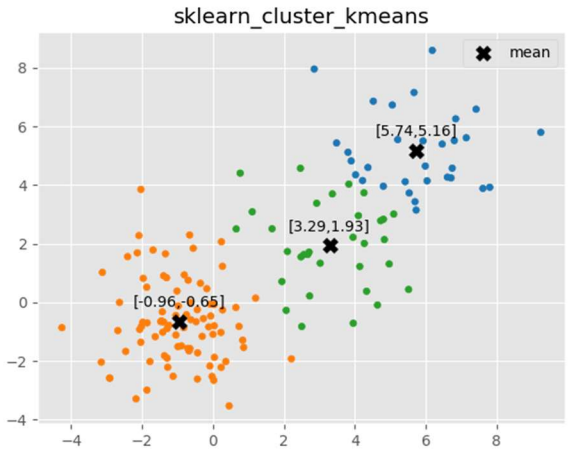
[-0.01510904 1.30286616]]

Part II: Software Familiarization

The libraries using K-means and Expectations Maximization algorithm we found are “sklearn.cluster.KMeans” and “sklearn.mixture.GaussianMixture”.

(1) K-means

The part1 k-means algorithm generates similar outcomes from sklearn library when k equal to 3. The graph is as follows.



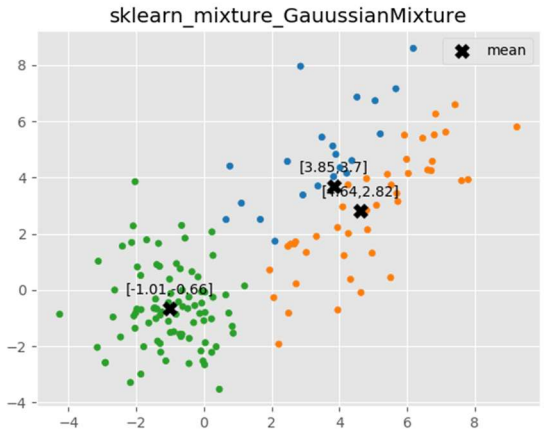
There are many parameters used to create clusters through KMeans library. Among them, we found that we could work on ‘init’ and ‘n_init’ to optimize our codes. The method for initialization is chosen through the parameter

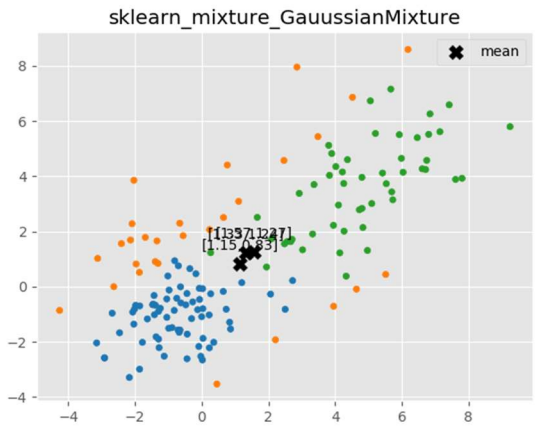
'init'. There are two methods, 'k-means++', 'random' and the default is set to be 'k-means++' which selects initial centroids in a smarter way to speed up convergence. 'random' means choose initial centroids randomly from data, which is also the way our k-means algorithm uses. We might be able to decrease the number of iterations if we can utilize 'k-means++'.

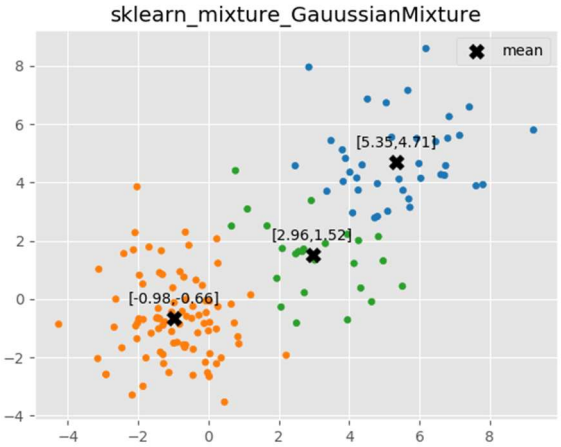
As the K-means algorithm guarantees only the local minimum solution, it is better that we run the code several times to get a chance to find a global solution. Therefore, the parameter 'n_init' is introduced. By setting the parameter, the KMeans library algorithm will run with different initial centroids and return the best output. We have already optimized our algorithm by adding the parameter. So comparing our plot and the package plot, the results are similar. There are some slight differences in classifying points in the middle of two centroids. This is unsolvable because K-Means do not have soft clustering.

(2) Expectation Maximization algorithm (GMM)

Besides parameter 'init' and 'n_init', GaussianMixture library also includes parameter 'init_params', offering the methods to initialize the program. The possible methods are 'kmeans' and 'random'. While 'random' chooses the random points (the same as what we did in our EM algorithm), 'kmeans' estimate the random points using k-means algorithm. Under this data set, the results of using 'random' are shown as below. They are not consistent, and one of them seems to not have explainable clusters.







Nonetheless, if we set the parameter to be ‘kmeans’, then the result (shown as below) is more similar to what we got from kmeans algorithm and more stable. In other words, we can get similar outcomes after each run of the program. Therefore, this supports our optimization of using centroids from K-Means. Comparing our plot with sklearn’s, ours have a little bit noisy points. This might be caused by the iteration time. After we add to higher iteration time, the results look better. One or two odd points are acceptable.

Part III: Applications

(1) K-Means:

Kmeans algorithm has been applied in many areas and one of them is image segmentation. Image segmentation classifies the image into different clusters according to its colors. The application helps recognizing the similar attributes. Take autonomous vehicles for example, the machine can detect the surroundings, recognize the objects as cars, passengers or other things and do the corresponding actions. Additionally, K-means is applied to the healthcare industry. The algorithm makes it possible to identify cancer cells in the early stage and therefore save people’s lives.

(2) Expectation Maximization algorithm:

Besides clustering, EM algorithm is used to estimate parameters of Hidden Markov Models (HMM). As there are two parameters in the maximum likelihood estimation, it is difficult to find a simple way using any iterative methods to derive the parameters. However, we can use the EM algorithm just as we do for the clustering problems. As a result, we can analyze part-of-speech tagging (POS tagging) which is beneficial to generate sentences with more accurate grammar and words.