Prathik Sannecy
Machine Learning
Programming Assignment 1
2/16/2020

Running the Code

The code is written in Python3. It uses the numpy, matplotlib, and scipy libraries that need to be installed (not installed by default). To run the code, go to the Source directory.

K-Means

To run K-Means algorithm, enter the following at the terminal:

```
python3 kMeans.py
```

The output of this code is the sum of squares error for the model(s), where each model consists of k clusters. Furthermore, it displays a scatterplot of the clustering of the model that had the least error (or just of the model itself, if only one k value is specified).

GMM

To run GMM algorithm, enter the following at the terminal:

```
python3 GMM.py
```

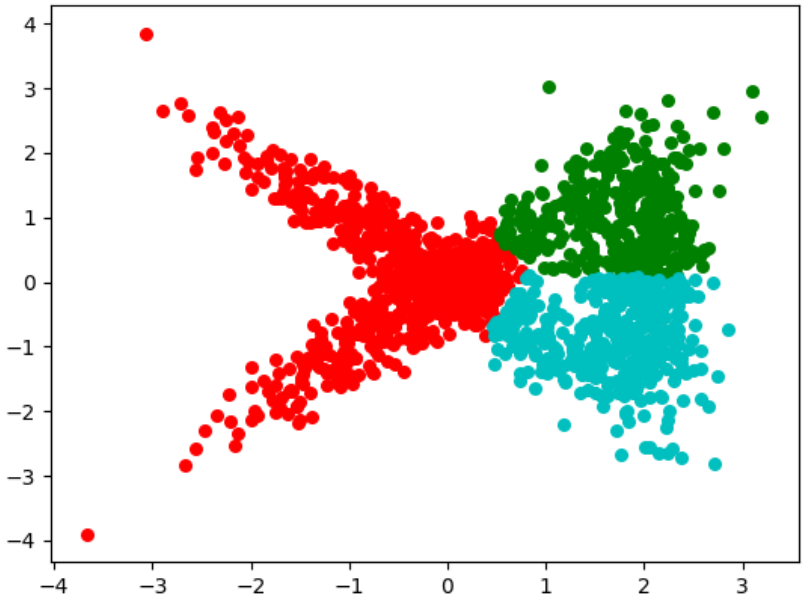
This code uses functions from kMeans.py , so GMM.py must be stored in the same directory. The output of this code is the calculated parameters for the Gaussians for the model(s), where each model consists of k clusters. Furthermore, it displays a scatterplot of the clustering of the model that had the highest likelyhood (or just of the model itself, if only one k value is specified).

Note: This program takes a long time to run (minutes), especially for large values of k

K-Means Approach/Outputs

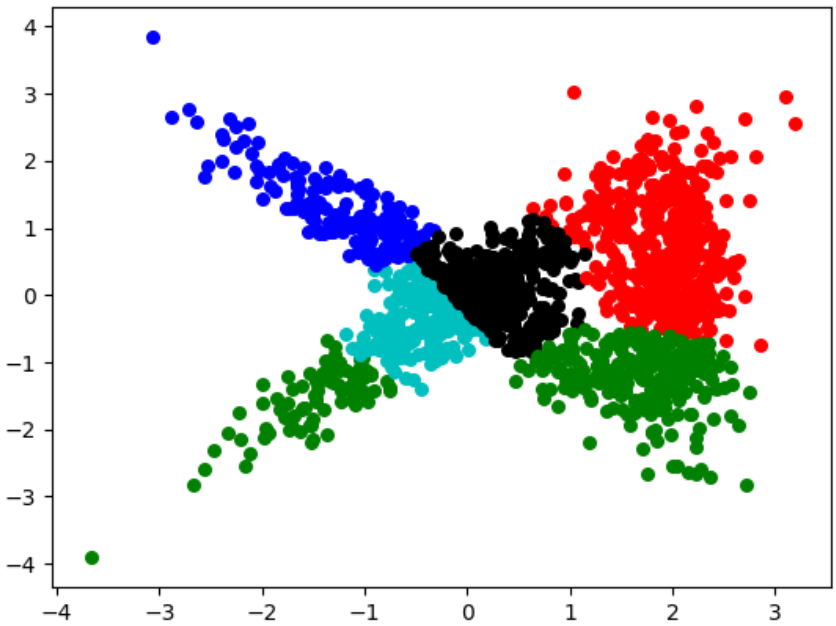
I ran K-Means algorithm on the GMM_dataset 546.txt from k = 2 to k = 7. Below is a report of the sum of squares error for each of these models, as well as a few example scatterplots of the clustering produced:

```
number of clusters: 2; clustering error:2228.619218804034
number of clusters: 3; clustering error:1539.2566749916375
```



K = 3 for K-Means

```
number of clusters: 4; clustering error:1103.5070477213308
number of clusters: 5; clustering error:773.1326716983122
number of clusters: 6; clustering error:626.898558509326
number of clusters: 7; clustering error:552.165232120474
```



K = 7 for K-Means

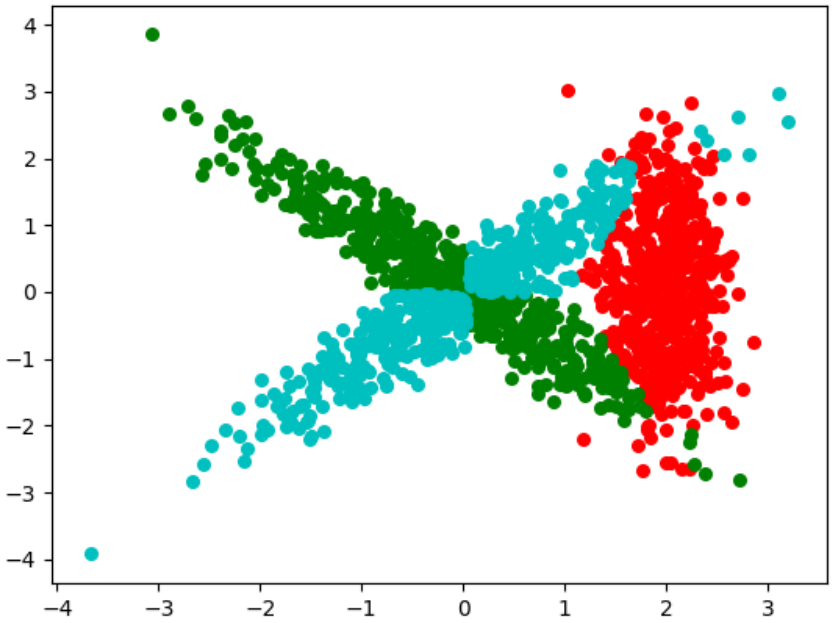
The approach I used was to run the algorithm 10 times per k value (this is parameterized and can be changed), and chose the resulting model that had the lowest error.

Furthermore, I also parameterized the number of clusters to use. To run this algorithm on a certain number of clusters, change the k_values variable (e.g. 3 clusters means k_values = [3]; to run on cluster size 3 and cluster size 4, k_values = [3, 4]).

GMM Approach/Output

I ran GMM algorithm on the GMM_dataset 546.txt from k = 2 to k = 7. Each model was initialized by first running K-Means for 1 iteration. Below is a report for the log likelyhood for each of these models, the clusters' centroids, covariances, and pi value, as well as a few example scatterplots of the clustering produced:

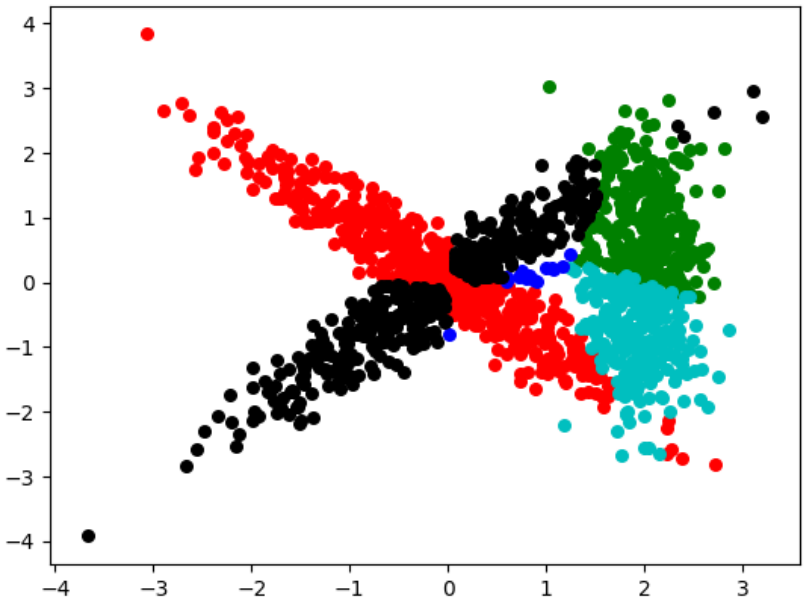
```
k = 2
        log likelyhood = -4405.535639902023
        cluster 0
                centroid =  [0.8886072001256571, 0.06184054451341702]
                cov =  [[1.5053019730155965, 0.6076830191802947], [0.6076830191802947, 0.9897878560999538]]
                pi =  0.6819002825317887
        cluster 1
                centroid =  [0.026940295879774066, 0.04340202709892162]
                cov =  [[1.4034449725810922, -1.2583045939536177], [-1.2583045939536177, 1.246622623641273]]
                pi =  0.31809971746821053

k = 3
        log likelyhood = -3927.4020073710885
        cluster 0
                centroid =  [1.9651601945969188, 0.10106993192594803]
                cov =  [[0.10257197014461489, -0.012502752463100963], [-0.012502752463100963, 1.0947476339563311]]
                pi =  0.3274221165187859
        cluster 1
                centroid =  [-0.08463132831053405, 0.10900604224293464]
                cov =  [[1.060878074417794, -0.9900448534706775], [-0.9900448534706775, 1.0376371633824615]]
                pi =  0.34086142054146196
        cluster 2
                centroid =  [-0.0002356050981812372, -0.04302839833244013]
                cov =  [[1.0535545013547944, 0.9997315824383876], [0.9997315824383876, 1.0688338847257588]]
                pi =  0.3317164629397514
```



K = 3 for GMM

```
k = 4
        log likelyhood = -3924.199108367878
        cluster 0
                centroid =  [0.00897763637958775, -0.03669253766270188]
                cov =  [[1.0536931484114818, 0.9960562011706715], [0.9960562011706715, 1.0630000167539522]]
                pi =  0.33652684411975037
        cluster 1
                centroid =  [-0.13385869753528104, 0.14476030854647046]
                cov =  [[1.0225887962352949, -0.9687525504760887], [-0.9687525504760887, 1.0284938277047215]]
                pi =  0.3277822917626746
        cluster 2
                centroid =  [1.8498129191169017, -0.7610626785936563]
                cov =  [[0.1178739156830891, -0.056136103989261915], [-0.056136103989261915, 0.5379080601625922]]
                pi =  0.13240593245647608
        cluster 3
                centroid =  [2.019039426905894, 0.5983848786386368]
                cov =  [[0.09111959690270226, -0.054135945679752245], [-0.054135945679752245, 0.7468278081847908]]
                pi =  0.20328493166109846
```

```
k = 5
        log likelyhood = -3923.0893551945983
        cluster 0
                centroid =  [0.059360902065327854, -0.10748527920608786]
                cov =  [[0.3355125001760375, 0.187812723697711 18], [0.18781272369771118, 0.2261916540797536]]
                pi =  0.08086945774163572
        cluster 1
                centroid =  [-0.04832029580568827, -0.06471788537158064]
                cov =  [[1.1665141084016084, 1.1404552075760914], [1.1404552075760914, 1.2264419630086516]]
                pi =  0.26174638108352605
        cluster 2
                centroid =  [1.9290089714350158, -0.6575431823986454]
                cov =  [[0.10404974105004246, -0.01443887641400273], [-0.01443887641400273, 0.5450738323362375]]
                pi =  0.15966444651688724
        cluster 3
                centroid =  [-0.12468379956224794, 0.13794830650279655]
                cov =  [[1.0790240349171383, -1.0253633450512145], [-1.0253633450512145, 1.0842868204801075]]
                pi =  0.31853459487555696
        cluster 4
                centroid =  [1.9760588055141368, 0.796116141739684]
                cov =  [[0.10680551109175215, -0.03915546885217868], [-0.03915546885217868, 0.6266171205120464]]
                pi =  0.1791851197823939
```



K = 5 for GMM

```
k = 6
        log likelyhood = -3918.0386707392804
        cluster 0
                centroid =  [1.235368598276353, -1.0584467567431521]
                cov =  [[0.24244088496731536, -0.124359089679625 74], [-0.12435908967962574, 0.186339274857412 08]]
                pi =  0.05443383087173869
        cluster 1
                centroid =  [-0.585332284543592, -0.5892323666113525]
                cov =  [[0.7035617803424422, 0.6926280472106192], [0.6926280472106192, 0.7934936661641477]]
                pi =  0.17283549333468196
        cluster 2
                centroid =  [1.4157617576349781, 1.3620395291915883]
                cov =  [[0.39391194787705736, 0.2932092122442728], [0.2932092122442728, 0.336290723524283 24]]
                pi =  0.07973236891046148
        cluster 3
                centroid =  [-0.33869415741135084, 0.3474066264332051]
                cov =  [[0.9572395416977993, -0.9077956295226809], [-0.9077956295226809, 0.9661624811683954]]
                pi =  0.2621589137180808
        cluster 4
                centroid =  [1.9798387607724448, 0.055782278139077 15]
                cov =  [[0.09744595054816438, -0.02174276214055791], [-0.02174276214055791, 1.052222705384212]]
                pi =  0.3066546950451071
        cluster 5
                centroid =  [0.1386189492933585, -0.010866424605860498]
                cov =  [[0.2251805844751402, 0.12299500675400293], [0.12299500675400293, 0.186184992729391 37]]
                pi =  0.12418469811993016

k = 7
        log likelyhood = -3916.690344948905
        cluster 0
                centroid =  [1.3531585791181557, 1.2708731674505558]
                cov =  [[0.4189564473407132, 0.32594923623113214], [0.32594923623113214, 0.386648949598382 86]]
                pi =  0.09061432410453092
        cluster 1
                centroid =  [0.10443226445344259, -0.024456205152308258]
                cov =  [[0.14515852227184783, 0.031978481175070496], [0.031978481175070496, 0.113542159284974 55]]
```

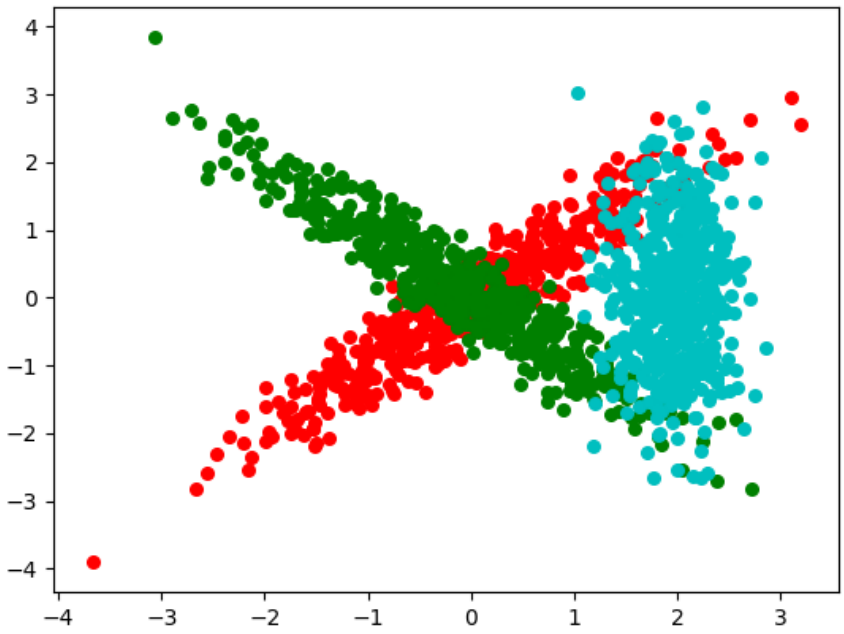
```
        pi =  0.13587061550390592
cluster 2
        centroid =  [1.976796099361479, 0.008919579619535465]
        cov =  [[0.09895534245155159, -0.01821349732616102], [-0.01821349732616102, 1.073011701344681]]
        pi =  0.3132692117498981
cluster 3
        centroid =  [-0.6531352103747847, 0.7419010339138904]
        cov =  [[0.15390436813520486, -0.1386232809397268], [-0.1386232809397268, 0.2259335394108048]]
        pi =  0.08840663408720605
cluster 4
        centroid =  [0.8568882855939821, -0.8130344568273554]
        cov =  [[0.3370869047928749, -0.2709913987385444], [-0.2709913987385444, 0.3250520070732728]]
        pi =  0.12455305418246523
cluster 5
        centroid =  [-1.549927278430198, 1.4148305634230063]
        cov =  [[0.36481600294016464, -0.39686000369853947], [-0.39686000369853947, 0.5172757993723079]]
        pi =  0.057151056585280934
cluster 6
        centroid =  [-0.5363005804508926, -0.5461249069155184]
        cov =  [[0.7033631355707649, 0.6874841752182766], [0.6874841752182766, 0.7863034158194406]]
        pi =  0.19013510378671217
```

The approach I used was to run the algorithm 5 times per k value (this is parameterized and can be changed), and chose the resulting model that had the highest log-likelyhood.

Furthermore, I also parameterized the number of clusters to use. To run this algorithm on a certain number of clusters, change the k_values variable (e.g. 3 clusters means that k_values = [3]; to run on cluster size 3 and cluster size 4, k_values = [3, 4]). I also added a tolerance parameter and a maximum number of iterations parameter to help the algorithm converge.

For generating the clusters in the scatterplots, I used the gamma function. Since the gamma function tells how much "responsibility that component k takes for “explaining” the observation x”, for each point, I found which cluster had the largest gamma function output for that particular point; I then assigned that point to that cluster.

Results and Conclusions



Original 3-Gaussian Dataset

As we can see above, using the GMM models is able to more tightly fit the data than the K-Means models. This is because we can use ellipses for clustering the data, rather than circles, which gives more flexibility. In the scatterplots above, we can see that the K-Means algorithm simple grouped together points that were close together. The GMM algorithm, on the other hand, was able to recognize eliptical shapes and cluster data points in that fashion. Due to this reason, the GMM algorithm was able to match with the actual 3-Gaussian dataset (above).

Another thing to note is that as the number of clusters increases, our models are able to more tightly fit the data for both GMM models as well as K-Means models. For GMM models, this is represented by the increase of the likelyhood, and for the K-Means models, this is represented by the decrease of the error.

Additionally, as we may expect, the GMM algorithm took much more time in general to run than the K-Means algorithm, as there were more computational steps required; this can be mitigated by adding some tolerance and putting a hard limit on the number of iterations, but it still isn't as fast as K-Means.

Finally, I noticed that as the number of clusters increased, both K-Means and the GMM algorithms took longer to run. This is to be expected, as both of these algorithms depend on comparing individual data points to the different clusters.