INF552 Machine Learning for Data Informatics HW2

1. Contribution:

a. Shih-Yu Lai (50%):

Email: shihyula@usc.edu

USC-ID: 7183984563

Implement K-Means code and write the ReadMe.

b. Shiuan-Chin Huang (50%):

Email: shiuanch@usc.edu

USC-ID: 9815633745

Implement EM+GMM code and write the ReadMe.

2. Steps of the algorithm

In Kmeans.py, we store the data and the centroid in the array. After we get the data from cluster.txt, we start to find K centroids. Then, calculate the distance of all the points to centroids and classify them. After this, we recompute new centroid for each cluster. Repeat classify all the points and recompute the centroid until K centroids won't change.

In GMM.py, we use the result from K means to calculate the initial mean, amplitude and covariance. Using these parameters in function Maximization as input to calculate gamma. Then, using the gamma as input parameter in function Expectation to calculate the mean, amplitude and covariance for the next run. Repeating these two steps until the parameters converge.

We convert the input data from list to array so that it is easier calculate with NumPy.

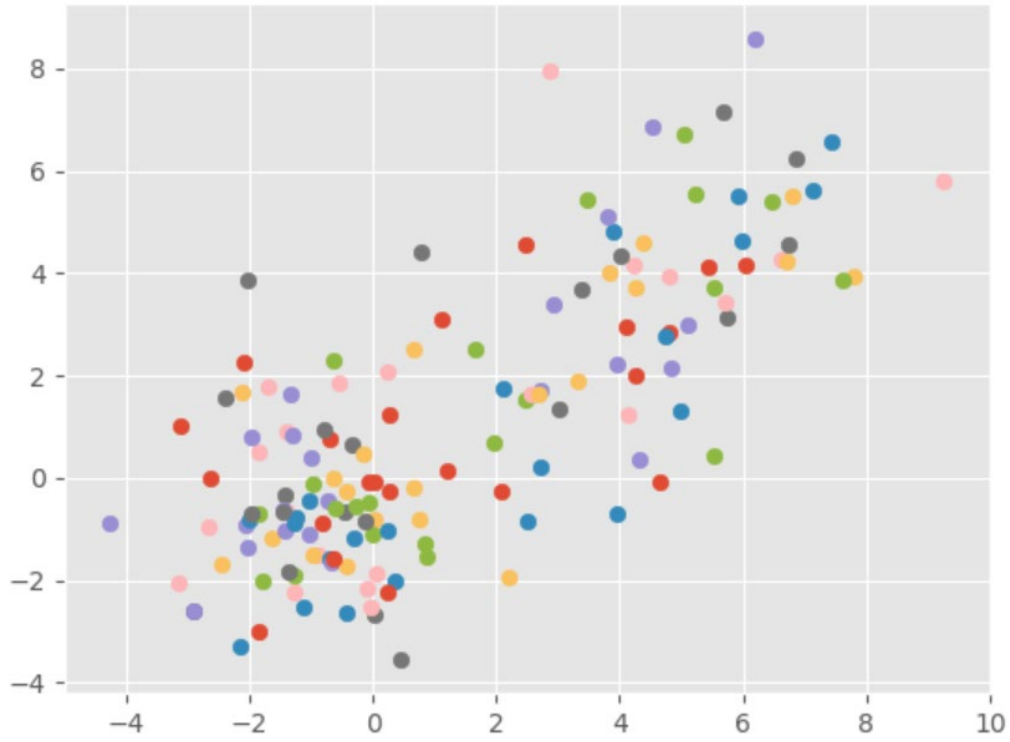
3. Our Outcome

In Kmeans.py, we will draw the centroids and the K clusters.

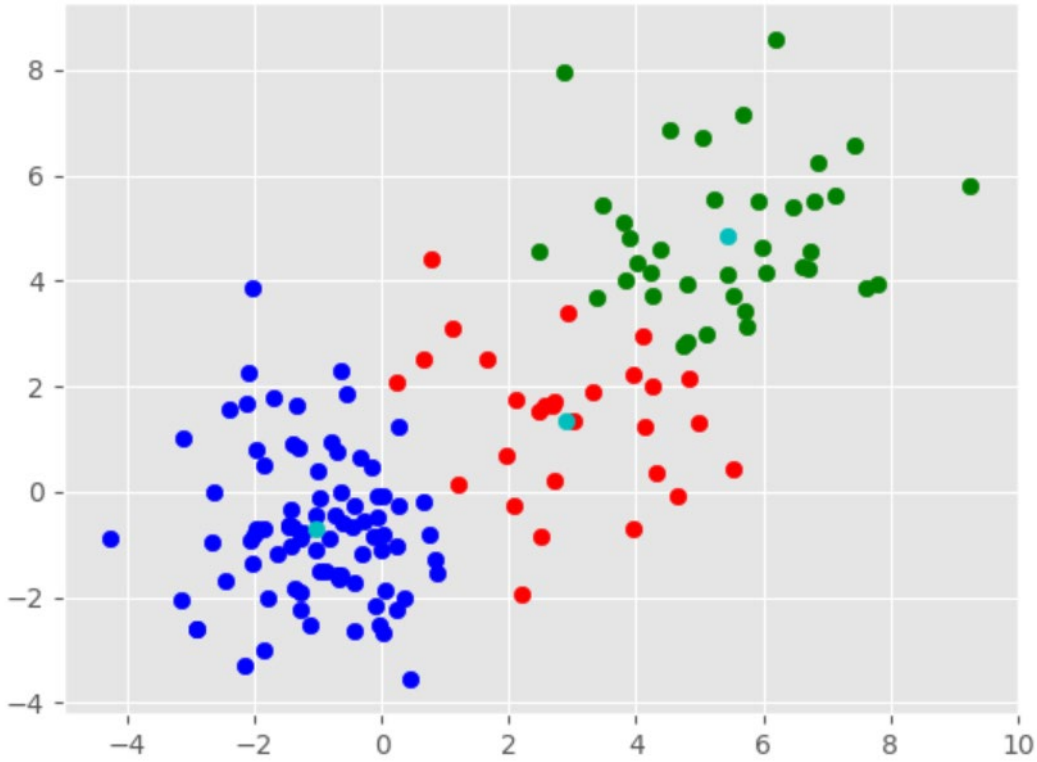
In GMM.py, we will report the mean, amplitude and covariance matrix of each Gaussian in GMM.

K-means:

Original data from clusters.txt.



Print out the 3 centroids(Cyan) and three clusters.



GMM:

```
Mean 1 :
[3.2810867354954234, 1.9490232918291774]
Covariance 1 :
[2.097060784373965, 0.38268374058274135]
[0.38268374058274135, 2.8655230983255824]
Amplitude 1 :
0.2271679495518719
Mean 2 :
[-0.9827116147452106, -0.6391826385328382]
Covariance 2 :
[1.1962430171129825, -0.09490339050213617]
[-0.09490339050213617, 2.0223491207509237]
Amplitude 2 :
0.565177470352145
Mean 3 :
[5.628466142007521, 4.994586468062794]
Covariance 3 :
[2.264360341853474, 0.1883247744512322]
[0.1883247744512322, 2.1511922191109027]
Amplitude 3 :
0.20765458009598306
```

Output of the mean, amplitude, and covariance matrix of each Gaussian in GMM

4. Challenge

To implement the k-means, we need to pick the random centroid first. But how to pick a good centroid is a question. Also, if we pick all the centroids randomly, then we may pick two centroids really near to others. In this case, the final result may be bad.

Hence, in our code, we randomly pick the first centroid from the data. Then we calculate the distance of other points to this centroid, try to find the farthest one or not near this centroid. Same with the third centroid. And we want to avoid the first centroid and the third centroid are near to each other, so we will multiply the random weight to separate them.

The concept of GMM is not difficult; however, there are lots of symbols so that it takes some time to understand the complex equations. Undoubtedly, it

also takes lots of time to think about how to put these mathematical functions into practice, especially doing the matrix calculations in coding. NumPy helps a lot with matrix calculations.

5. Software Familiarization

Scikit-learn K-means:

There are some parameters in this classifier we can adjust conveniently when the data set is too large or there are too many dimensions of features. For instance,

n_clusters: The number of clusters to form as well as the number of centroids to generate. (default : 8)

max_iter: Maximum number of iterations of the k-means algorithm for a single run. (default : 300)

init: {'k-means++', 'random' or an ndarray}

precompute_distances: {'auto', True, False}. True : always precompute distances. False : never precompute distances

In our code, the way we choose the centroids is really similar to the kmeans++, but is simpler than kmeans++. If we want to improve our code, we can choose the centroids like kmeans++ or we can split our data into K areas and pick the centroids from them. Also, we can use other methods to help us find the best result like using the elbow method to find the best K.

Scikit-learn.Mixture.GMM

There are some parameters in this classifier you can initialize when starting the calculations which provides a more correct estimation. For instance,

min_covar: Floor on the diagonal of the covariance matrix to prevent overfitting. (defaults : 1e-3)

tol: Convergence threshold. EM iterations will stop when average gain in log-likelihood is below this threshold. (default: 1e-3)

init_params: Controls which parameters are updated in the initialization process. Can contain any combination of ‘w’ for weights, ‘m’ for means, and ‘c’ for covars. (defaults: ‘wmc’)

In our code, we check if the EM iteration should stop based on both the maximum iteration time and converge. If the mean value difference is less than a specific amount, then we set converge = True. If we want to improve our code, we can compute the log probability of data points, and check if it is below the threshold to decide convergence. In addition, if we want to modify our code to prevent overfitting, we can also add one more parameter like min_covar in this library.

6. Applications

K-Means:

K-means is a versatile algorithm that can be used for any type of grouping. For example, like behavioral segmentation:

1. Segment by purchase history.
2. Segment by activities on application, website, or platform
3. Define personas based on interests.
4. Create profiles based on activity monitoring.

Or sorting sensor measurements:

1. Detect activity types in motion sensors.
2. Group images.
3. Separate audio.
4. Identify groups in health monitoring.

GMM:

1. Speaker recognition

Extracting features such as pitch, amplitude and frequency from those speech signals, and using GMM to identify a person.

2. Background Subtraction

Using color and depth information to detect moving objects based on GMM, which is a popular method in computer vision.

3. Image Segmentation

Based on GMM, we can calculate the probability of each pixel, then classify them to different category.