# Part 1: Implementation

The code is in ***Jiashi_Chen_Homework_2_Code.ipynb*** file.

I use **two-dimensional arrays** to save the coordinates of the centroids in K-means. I use **arrays from the numpy package and matrices from the pandas package** to save the parameters of the Gaussian distribution in GMM algorithm.

Code-level optimizations I perform is that I try to **use matrixes to calculate data instead of loops**, increasing calculation efficiency. I also use functions to modularize the calculation process, which is helpful for debugging and management.

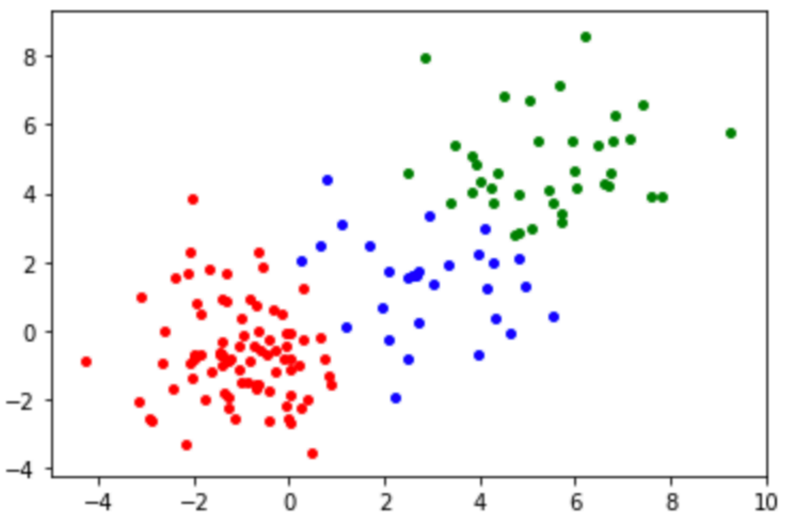
Challenges I face is that at the beginning I was a bit confused about the operation of matrices and arrays which leads to many wrong result.

Since each 3 centroids in K-means and weights in GMM are random at beginning, the output is not always the exactly same, however they are always close to other runs.
One of the **output** in **K-mean**s:

```
In [54]: K_means(data, 3)

Out[54]: [[5.738495346032257, 5.164838081193548],
          [-0.9606529070232559, -0.6522184128604652],
          [3.2888485605151514, 1.9326883657575762]]
```

**Visualized** the assignments:

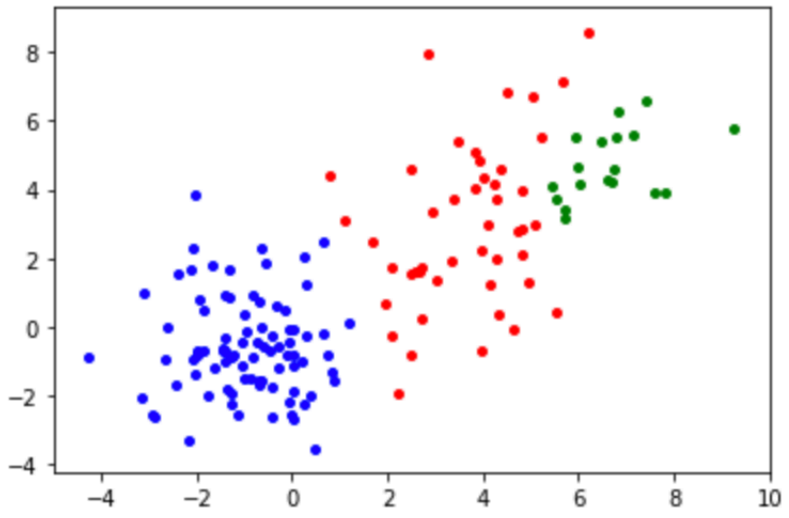


One of the **output** in **GMM**:

```
Mu, Sigma, Pi = GMM(data, 3)
print("The list of mean: ")
print(Mu)
print("----------")
print("The list of covariance matrix: ")
print(Sigma)
print("--------")
print("The list of amplitude: ")
print(Pi)
```

```
The list of mean:
[[3.73099187120709, 2.9855455782775535], [-0.9363099371558942, -0.5785580959583669], [6.155129816334989, 4.38410743
7555044]]
----------
The list of covariance matrix:
[matrix([[2.52092325, 1.69173506],
        [1.69173506, 5.88450297]]), matrix([[ 1.20167293, -0.05148645],
        [-0.05148645,  2.06069071]]), matrix([[1.83092718, 0.92832769],
        [0.92832769, 1.29369472]])]
--------
The list of amplitude:
[0.31599382 0.5657184  0.11828778]
```

**Visualized** the assignments:



The output of the two algorithms are similar. However, due to the difference between randomness and the algorithms themselves, the output has a certain difference.

## Part 2: Software Familiarization

**Sklearn package** offers a good implementation of the decision tree learning algorithm.

```
from sklearn.cluster import KMeans  ## obtain necessary package
from sklearn import mixture

estimator = KMeans(n_clusters=3)    ## K-means algorithm
estimator.fit(data)

modele = mixture.GaussianMixture(n_components=3, covariance_type='full')    ##GMM
modele.fit(data)
```

I used this package to fit the data in ***Jiashi_Chen_Homework_2_Code.ipynb*** file too.

### Comparsion and Improvement

I found that when computing the probability of each sample from a Gaussian function, we need to input the samples into the corresponding Gaussian probability density function. The calculation of the Gaussian probability density function requires the values of the inverse matrix and the determinant. The sklearn package **uses cholesky** to improves the efficiency a lot.

GaussianMixture has different options to constrain different types of estimated covariance: **spherical, diagonal, tied, full covariance.** So the model can be adjusted to fit more data.

Also the package considers many **exceptions** to make the code more applicable.

## Part 3: Applications

### K-means

K-means can help managers improve their customer base (working in their target area) and further segment customer categories based on customer purchase history, interest, or activity monitoring. Categorizing customers helps companies tailor specific advertisements to specific customer groups.

### GMM

GMM algorithm can achieve foreground segmentation. The GMM uses K (basically 3 to 5) Gaussian models to characterize the characteristics of each pixel in the image. After a new frame of image is obtained, the Gaussian Mixture Model is updated. Each pixel in the current image is used with the model. If model matching successfully, the point is determined as the background point, otherwise it is the foreground.