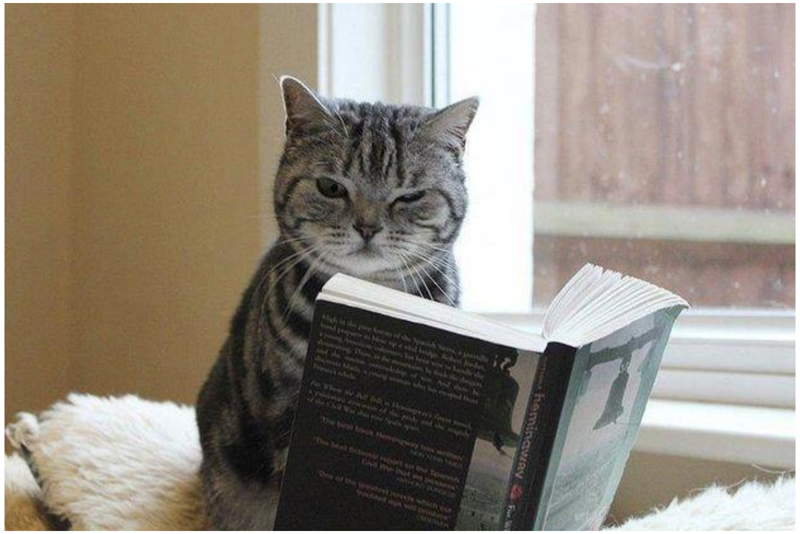
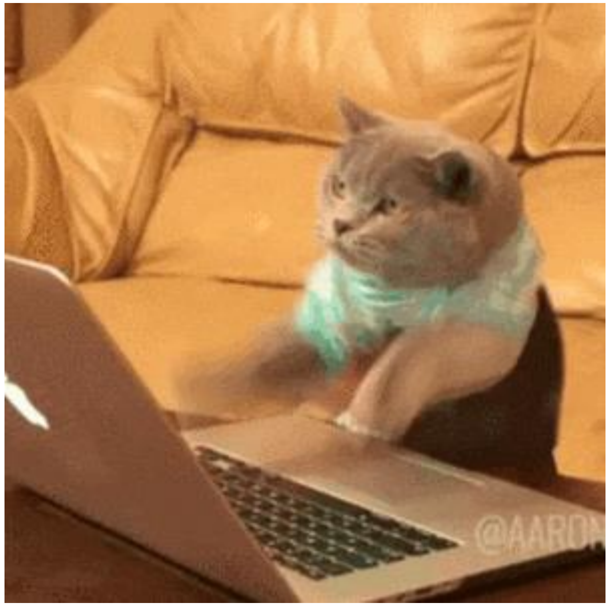
Python Practice for k-Means & GMM Clustering

Hanwool Jeong

hwjeong@kw.ac.kr

Coding Completes Your Understanding





Features of Figure

- Thus far, we plot graphs with plt.plot(X,Y, …) after the following import statement

```
import matplotlib.pyplot as plt
```

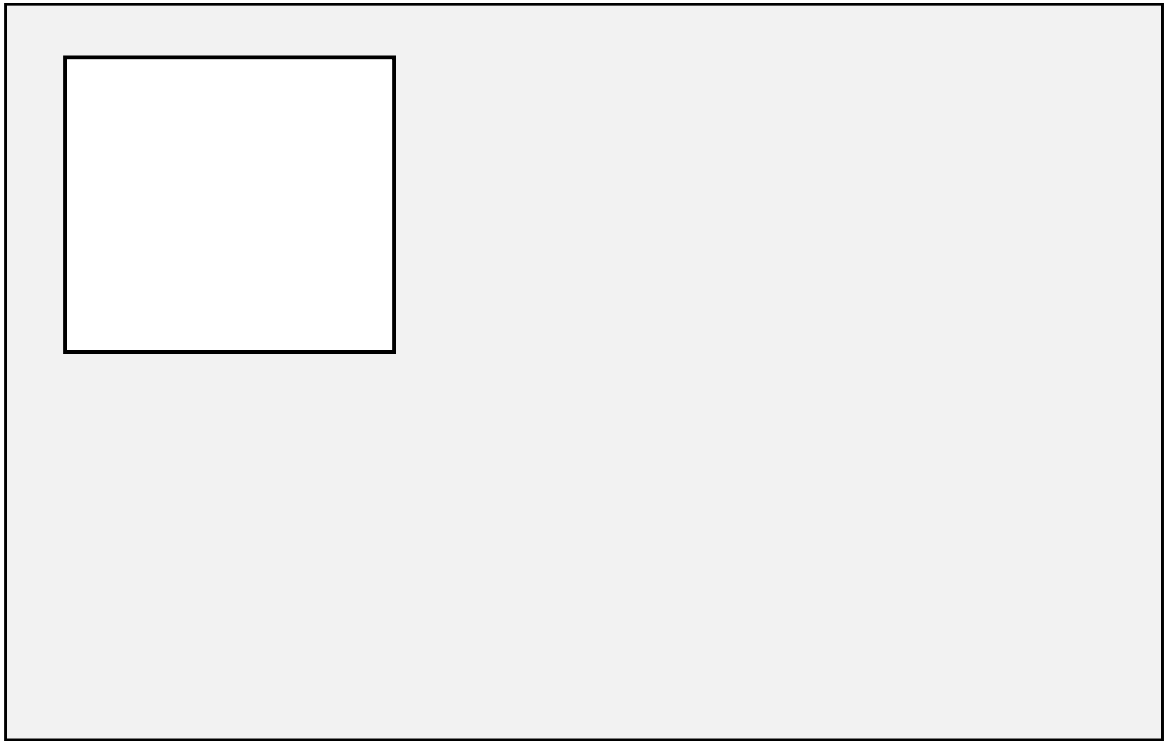
- The following usage of plt is more useful:

```
f1 = plt.figure(1)
ax1 = f1.add_subplot(111)
ax1.plot(x, y, 'b.')
ax1.plot(Z[:,0], Z[:,1], 'r*')
```

f1.add_subplot()

ax1 = f1.add_subplot(231)

ax1.plot(X, Y, “b”)



DataFrame in Pandas

- Structure : Data name & Index ➔ Database

Index	X	Y	K
0	-0.365734	0.673413	1
1	-0.384726	0.344068	1
2	-0.551603	0.482672	1
3	-0.68766	0.238392	1
4	-1.30458	1.46957	0
5	-0.78629	0.415307	0
6	-0.30152	0.22838	1

- **df[“Ac”]** : column indexing
- **df.iloc[“Ar”]** : row indexing
- What would be **df.iloc[4][“X”]**?
- We can convert X into dataframe by using pd.DataFrame(X)

Why is DataFrame Useful?

- For now, we will focus on Groupby
- What will happen if we run: df.groupby(“K”)

Index	X	Y	K
0	-0.365734	0.673413	1
1	-0.384726	0.344068	1
2	-0.551603	0.482672	1
3	-0.68766	0.238392	1
4	-1.30458	1.46957	0
5	-0.78629	0.415307	0
6	-0.30152	0.22838	1

← **df.columns**

Data used for Classification

- Data1:
  - https://raw.githubusercontent.com/hanwoolJeong/lectureUniv/main/ClassificationSample.txt
- Data2:
  - https://raw.githubusercontent.com/hanwoolJeong/lectureUniv/main/ClassificationSample2.txt

Importing Required Modules & Load Data

- Import modules:

```
import numpy as np
import math as m
import matplotlib.pyplot as plt
import pandas as pd
```

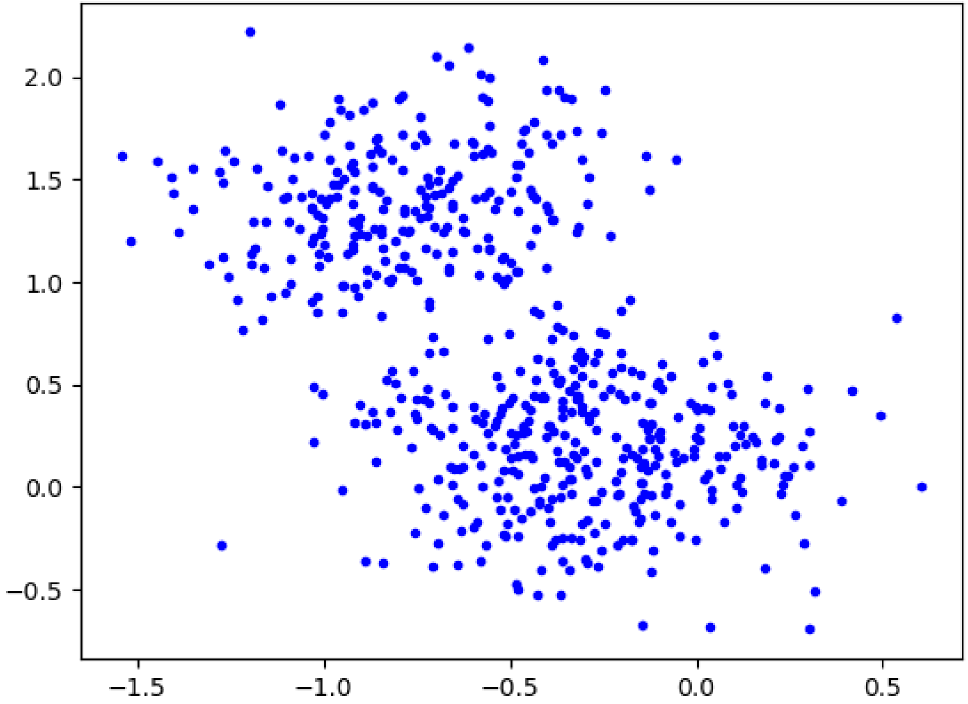
- Load data with pandas.read_csv

```
dfLoad =
pd.read_csv("https://raw.githubusercontent.com/hanwoolJeong/lectureUniv/main/ClassificationSample.txt", sep="\s+")
```

Plot for Checking Data

```
samples = np.array(dfLoad)
x = samples[:,0]
y = samples[:,1]

f1 = plt.figure(1)
ax1 = f1.add_subplot(111)
ax1.plot(x, y, 'b.')
```



Revisit Pseudo Code for k-Means Clustering

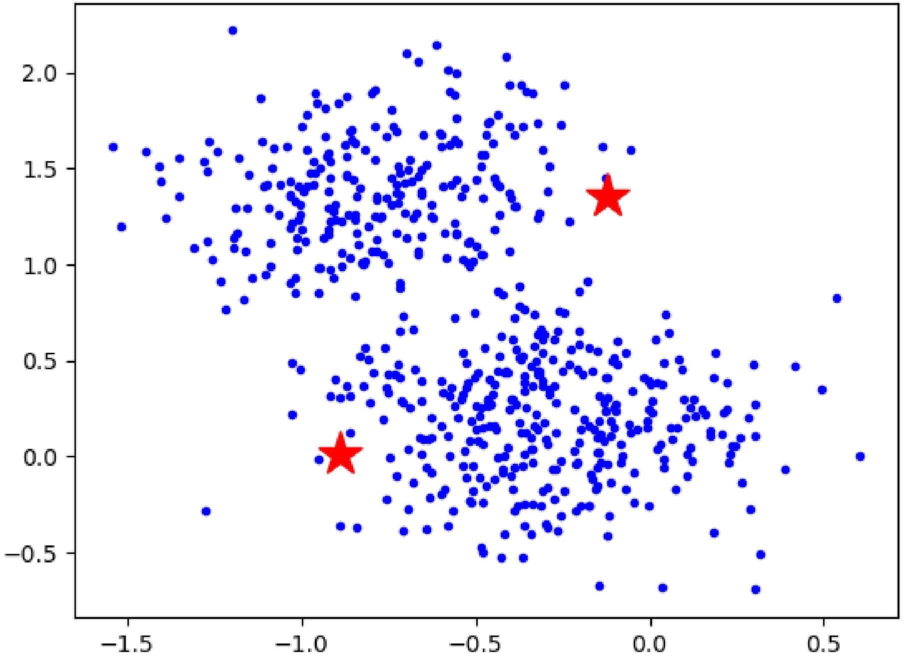
- Given inputs **X** and the number of clusters of **k**, K
- Defining latent variable matrix **Z**, algorithm is like as follows:

```
Initialize Z = {z_1, z_2, z_3, … z_K}
while(true)
        for(i=1 to N)
             Map x_i into the nearest z_j
        if(No change of mapping from the prev. loop) break
        for(j=1 to K)
             replace z_j with the mean of the x_i mapped to z_j
for (j=1 to K)
        allocate the samples mapped to z_j to k_j
```

- Output : $\mathbf{k} = \{\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_K\}$

Initialize Latent Variables

- How can we effectively initialize **Z** that represents the potential means of clusters? ➔ **Let's try to avoid too irrelevant or odd value ➔ How?**

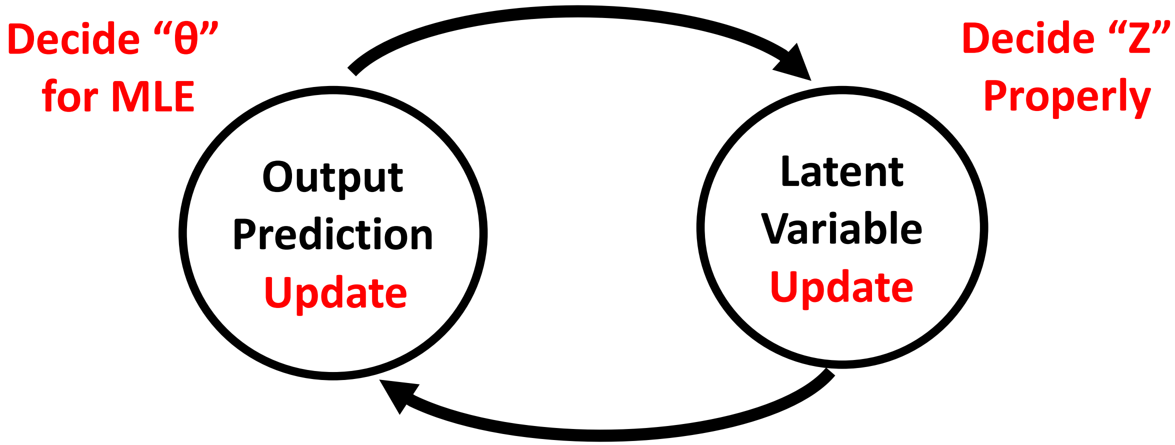



```
[mx, sx] = [np.mean(x), np.std(x)]
[my, sy] = [np.mean(y), np.std(y)]
z0 = np.array([mx+sx,my+sy]).reshape(1,2)
z1 = np.array([mx-sx,my-sy]).reshape(1,2)
Z = np.r_[z0, z1]
ax1.plot(Z[:,0], Z[:,1], 'r*')
```

While Loop for Update Cycle b/w Latent Variable / Output

- Revisit k-Means pseudo code: **Let's focus on a single loop first**

```
Initialize Z = {z_1, z_2, z_3, ... z_K}
    while(true)
        for(i=1 to N)
            Map x_i into the nearest z_j
        if(No change of mapping from the prev. loop) break
        for(j=1 to K)
            replace z_j with the mean of the x_i mapped to z_j
    for (j=1 to K)
        allocate the samples mapped to z_j to k_j
```



Output Update in Single Loop

You remember this?

```
samples = np.array(dfLoad)
x = samples[:,0]
y = samples[:,1]
```

- Note that we have data in **samples** as numpy.array

Initialize Z = {$\mathbf{z}_1$, $\mathbf{z}_2$, $\mathbf{z}_3$, … $\mathbf{z}_K$}
    while(true)
        for(i=1 to N)
            Map x_i into the nearest z_j
        if(No change of mapping from the prev. loop) break
        for(j=1 to K)
            replace $\mathbf{z}_j$ with the mean of the $\mathbf{x}_i$ mapped to $\mathbf{z}_j$
    for (j=1 to K)
        allocate the samples mapped to z_j to k_j

How can you code this?

- Express your goal verbally. There might be different ways coding it.

```
N = len(samples)
for i in range(N):
    k[i] = np.linalg.norm(samples[i,:]-Z[0,:]) > np.linalg.norm(samples[i,:]-Z[1,:])
```

Latent Variable Update in Single Loop

- Let's focus on Z update!

```
Initialize Z = {z_1, z_2, z_3, … z_K}
    while(true)
        for(i=1 to N)
            Map x_i into the nearest z_j
        if(No change of mapping from the prev. loop) break
        for(j=1 to K)
            replace z_j with the mean of the x_i mapped to z_j
    for (j=1 to K)
        allocate the samples mapped to z_j to k_j
```

- Again, express your goal verbally. **Can we use groupby?**

```
dfCluster = pd.DataFrame(np.c_[x, y, k])
dfCluster.columns = ["X", "Y", "K"]
dGroup = dfCluster.groupby("K")

for i in range(numK):
Z[i,0:2] = np.array(dGroup.mean().iloc[i])
```

Break Condition

- We can check whether np.array k changes

```
Initialize Z = {z_1, z_2, z_3, … z_K}
    while(true)
        for(i=1 to N)
            Map x_i into the nearest z_j
        if(No change of mapping from the prev. loop) break
        for(j=1 to K)
            replace z_j with the mean of the x_i mapped to z_j
    for (j=1 to K)
        allocate the samples mapped to z_j to k_j
```

- What is kold == k? Is there any other method that fit out goal?

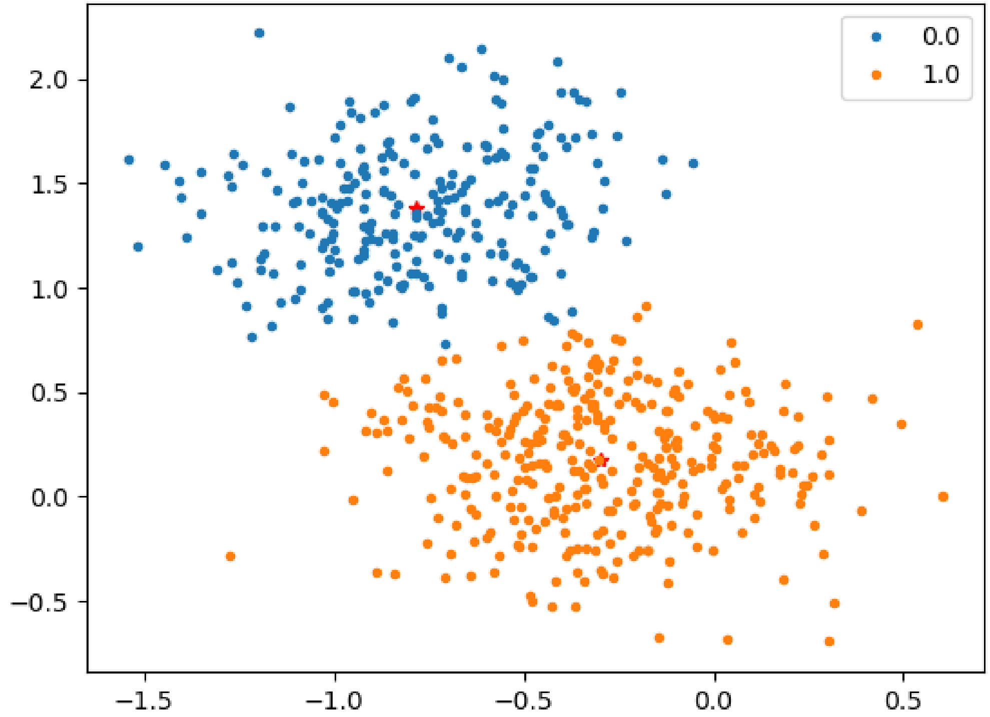
```python
kold = np.copy(k)

if np.alltrue(kold == k):
    break
```

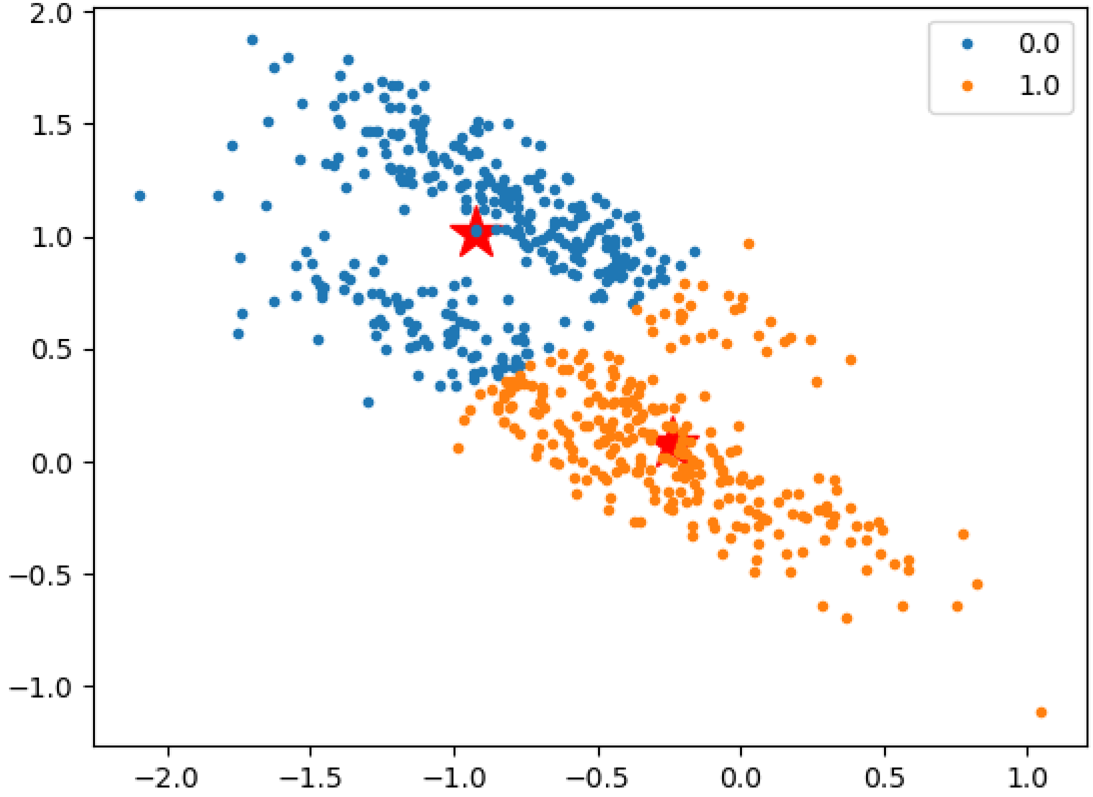
Visualize Result!

```
f2 = plt.figure(2)
ax2 = f2.add_subplot(1,1,1)
ax2.plot(Z[:,0], Z[:,1], 'r*')

for clusterName, group in dGroup:
    ax2.plot(group.X, group.Y, '.', label=clusterName)
ax2.legend()
```



How About This?



➔ **Let's apply GMM clustering!**

Revisit Key of GMM Clustering

- For E step,

$$r_{ik} \triangleq p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}) = \frac{p(z_i = k|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta})}{\sum_{k'=1}^{K} p(z_i = k'|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k', \boldsymbol{\theta})}$$

$$= \frac{\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k^{(t-1)})}{\sum_{k'} \pi_{k'} p(\mathbf{x}_i|\boldsymbol{\theta}_{k'}^{(t-1)})}$$

- For M step,

$$\pi_k = \frac{1}{N}\sum_i r_{ik} = \frac{r_k}{N}$$

$$\boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\mathbf{x}_i}{r_k}$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{r_k} = \frac{\sum_i r_{ik}\mathbf{x}_i\mathbf{x}_i^T}{r_k} - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T$$

GMM Clustering Pseudo Code

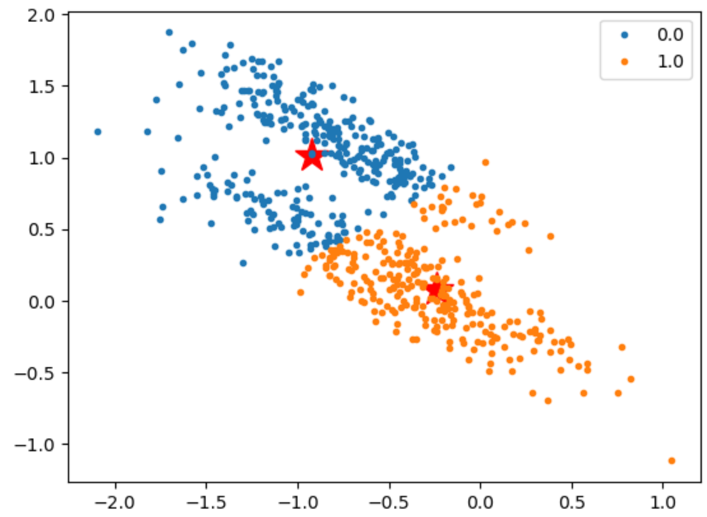
Initialize θ

while(until converge)

Estimate $\mathbf{r}_{ij}$ based on $\boldsymbol{\theta}$

Estimate $\boldsymbol{\theta}$ based on $\mathbf{r}_{ij}$

Initializing θ & r_{ij}




```
pi = np.ones(numK)*(1/numK)

[mx, sx] = [np.mean(x), np.std(x)]
[my, sy] = [np.mean(y), np.std(y)]

u0 = np.array([mx-sx,my+sy])
u1 = np.array([mx+sx,my-sy])
Sigma0 = np.array([[sx*sx/4, 0], [0, sy*sy/4]])
Sigma1 = np.array([[sx*sx/4, 0], [0, sy*sy/4]])

R = np.ones([N, numK])*(1/numK)
```

E step

$$r_{ik} \triangleq p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}) = \frac{p(z_i = k|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta})}{\sum_{k'=1}^{K} p(z_i = k'|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k', \boldsymbol{\theta})}$$

$$= \frac{\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k^{(t-1)})}{\sum_{k'} \pi_{k'} p(\mathbf{x}_i|\boldsymbol{\theta}_{k'}^{(t-1)})}$$

```
N0 = sp.stats.multivariate_normal.pdf(samples, u0, Sigma0)
N1 = sp.stats.multivariate_normal.pdf(samples, u1, Sigma1)

Rold = np.copy(R)
R = np.array([pi[0]*N0/(pi[0]*N0+pi[1]*N1), pi[1]*N1/(pi[0]*N0+pi[1]*N1)]).T
```

M Step

$$\pi_k = \frac{1}{N}\sum_i r_{ik} = \frac{r_k}{N}$$

$$\boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\mathbf{x}_i}{r_k}$$

You aware of r_k?

$$\boldsymbol{\Sigma}_k = \frac{\sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{r_k} = \frac{\sum_i r_{ik}\mathbf{x}_i\mathbf{x}_i^T}{r_k} - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T$$

```
pi = np.ones(N).reshape(1,N).dot(R)/N
pi = pi.reshape(2,)
weightedSum = samples.T.dot(R)
u0 = weightedSum[:,0]/sum(R[:,0])
u1 = weightedSum[:,1]/sum(R[:,1])
Sigma0 = samples.T.dot(np.multiply(R[:,0].reshape(N,1), samples))/sum(R[:,0]) - u0.reshape(2,1)*u0.reshape(2,1).
Sigma1 = samples.T.dot(np.multiply(R[:,1].reshape(N,1), samples))/sum(R[:,1]) - u1.reshape(2,1)*u1.reshape(2,1).
```