X:dim=10 n\_cluster=10

Experiment1.1:

The training data X was generated by the following code:

*from sklearn.datasets import make\_blobs*

*X, y = make\_blobs(n\_samples=10000, centers=10, n\_features=10,cluster\_std=0.4,*

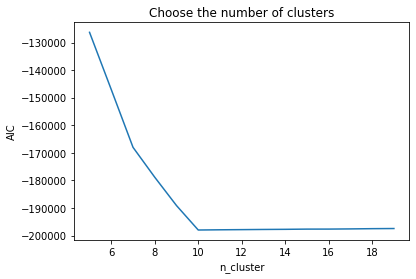
*random\_state=5)*

*import matplotlib.pyplot as plt*

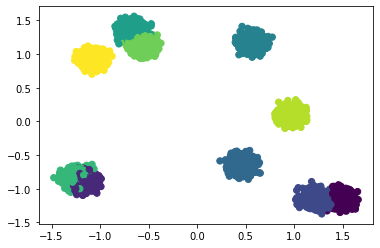
*X = (X - X.mean(axis=0)) / X.std(axis=0)*

It can be well clustered by GMM. The accuracy is 1.0.

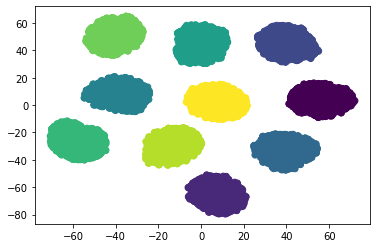
According to the AIC rule, the number of clusters should be 10.



On the first 2 dimension



Plot X\_embeded(T-sne,dim=2)



However, Variational gaussian mixture model suggests that the number of clusters is 8.

weights=[1.1752593e-18 1.1752593e-18 1.0000000e-01 1.0000000e-01 1.0000000e-01

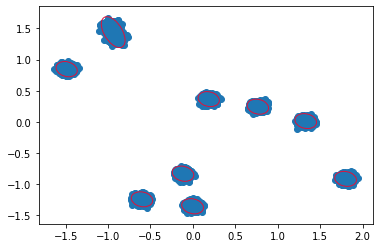
1.1752593e-18 1.1752593e-18 1.0000000e-01 1.0000000e-01 2.0000000e-01

1.1752593e-18 1.1752593e-18 1.1752593e-18 1.1752593e-18 2.0000000e-01

1.1752593e-18 1.1752593e-18 1.0000000e-01 1.1752593e-18 1.1752593e-18]

如果有的类特别近，别的特别远，就可能会失效

以2维为例，但可能调整初始化可以解决。



weights [7.42287514e-16 1.00000000e-01 1.00000000e-01 7.42287514e-16

7.42287514e-16 1.00000000e-01 1.00000000e-01 1.00000000e-01

1.00000000e-01 2.00000000e-01 7.42287514e-16 7.42287514e-16

7.42287514e-16 7.42287514e-16 7.42287514e-16 7.42287514e-16

1.00000000e-01 7.42287514e-16 1.00000000e-01 7.42287514e-16]

Experiment1.2:

The training data X was generated by the following code:

*from sklearn.datasets import make\_blobs*

*X, y = make\_blobs(n\_samples=10000, centers=10, n\_features=10,cluster\_std=1,*

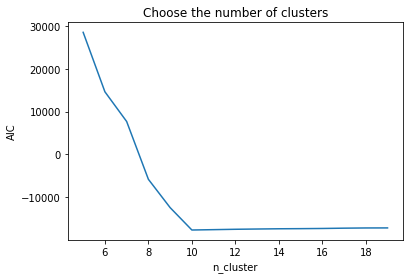
*random\_state=5)*

*import matplotlib.pyplot as plt*

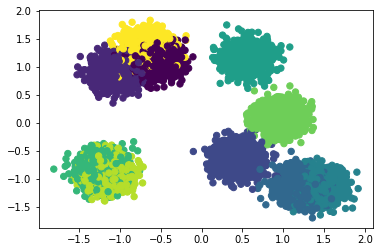
*X = (X - X.mean(axis=0)) / X.std(axis=0)*

It can be well clustered by GMM. The accuracy is 1.0.

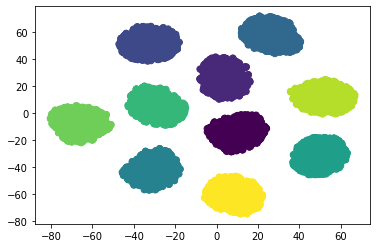
According to the AIC rule, the number of clusters should be 10.



On the first 2 dimension



Plot X\_embedded(T-sne,dim=2)



weights [2.49396153e-27 2.49396153e-27 1.00000000e-01 2.49396153e-27

1.00000000e-01 1.00000000e-01 1.00000000e-01 2.49396153e-27

2.49396153e-27 1.00000000e-01 2.49396153e-27 2.49396153e-27

1.00000000e-01 2.49396153e-27 1.00000000e-01 1.00000000e-01

1.00000000e-01 2.49396153e-27 1.00000000e-01 2.49396153e-27]

Experiment2：

*m\_true = np.random.randn(10, 10)*

*covs\_true = 0.05\*np.diag(np.ones(10))*

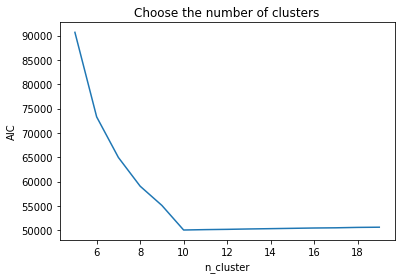
*X = np.concatenate([np.random.multivariate\_normal(m\_true[k], covs\_true, 1000) for k in range(len(m\_true))])*

*y = np.concatenate([k\*np.ones(1000) for k in range(10)])*

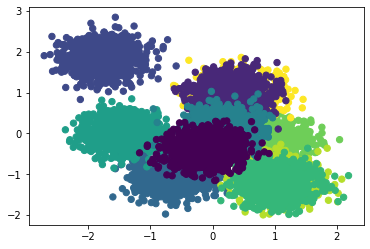
*X = (X - X.mean(axis=0)) / X.std(axis=0)*

It can be well clustered by GMM. The accuracy is 1.0.

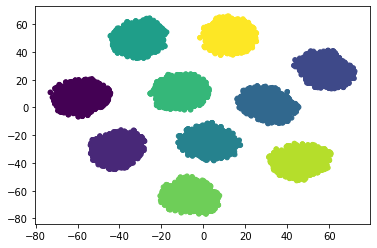
According to the AIC rule, the number of clusters should be 10.



On the first 2 dim



Plot X\_embedded(T-sne,dim=2)



Using VGMM we can get desired results:

weights [1.00000000e-01 9.18411035e-14 9.18411035e-14 9.18411035e-14

9.18411035e-14 1.00000000e-01 1.00000000e-01 1.00000000e-01

1.00000000e-01 9.18411035e-14 1.00000000e-01 9.18411035e-14

1.00000000e-01 1.00000000e-01 9.18411035e-14 9.18411035e-14

9.18411035e-14 1.00000000e-01 9.18411035e-14 1.00000000e-01]

X dim=100 n\_cluster=10

Experiment3

import numpy as np

*m\_true = np.random.randn(10, 100)*

*covs\_true = 0.05\*np.diag(np.ones(100))*

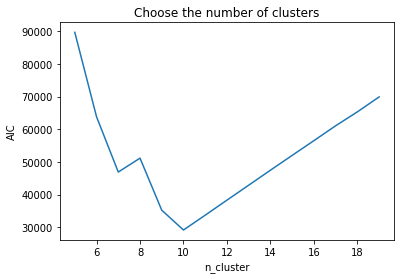
*X = np.concatenate([np.random.multivariate\_normal(m\_true[k], covs\_true, 1000) for k in range(len(m\_true))])*

*y = np.concatenate([k\*np.ones(1000) for k in range(10)])*

*X = (X - X.mean(axis=0)) / X.std(axis=0)*

It can be well clustered by GMM. The accuracy is 1.0.

According to the AIC rule, the number of clusters should be 10.



Using the diagonal version of vgmm(Sigma is diagonal) we get the desired results.

weights [0.1 0. 0. 0. 0.1 0.1 0.1 0. 0.1 0.1 0. 0.1 0. 0.1 0. 0. 0.1 0.

0.1 0. ]

Clustering accuracy=1.0