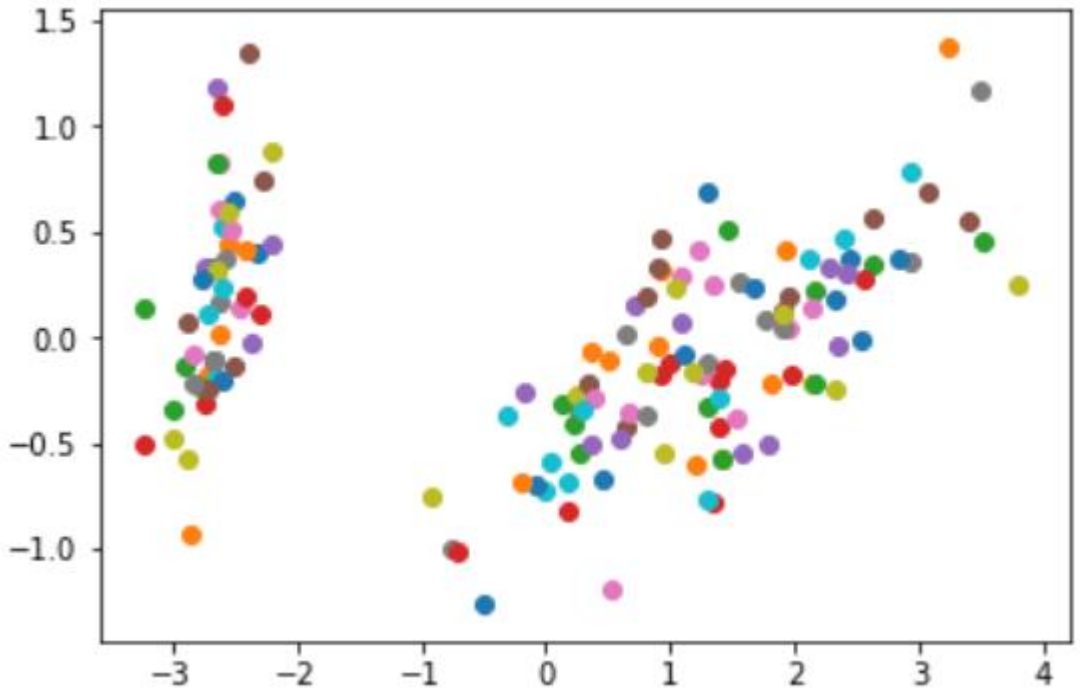
인공지능 과제5

20130940 장용훈

1-1 PCA분석

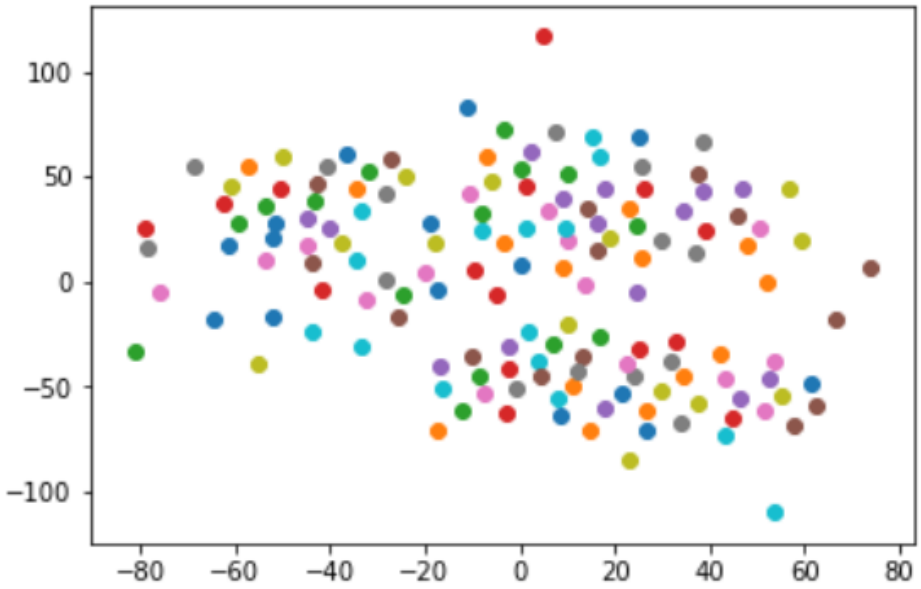
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from numpy import genfromtxt
import numpy as np
data = genfromtxt('data.csv', delimiter=',')
features = ['feature1','feature2','feature3','feature4']
data = data[1:]
pca = PCA(n_components=2)
X_r = pca.fit(data).transform(data)
plt.figure(32)
for x,y in X_r:
    plt.scatter(x,y)
plt.show()
```



2개의 부분으로 잘 나누어진 모습을 볼 수 있다. 하지만 약간의 잡음은 존재한다.

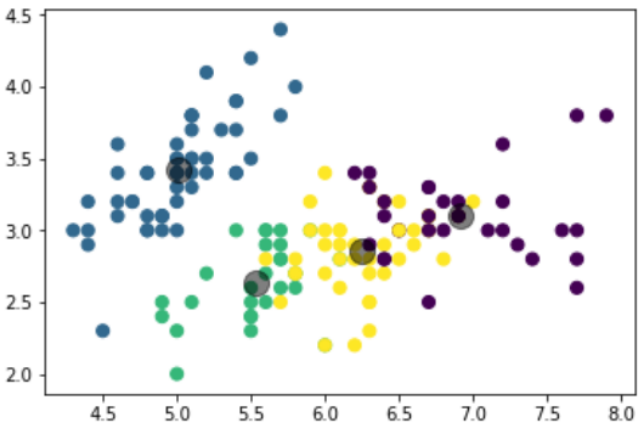
1-2 T-SNE 분석

```
#1-2 t-sne
from sklearn.manifold import TSNE
x_tsne = TSNE(n_components=2).fit_transform(data)
for x,y in x_tsne:
    plt.scatter(x,y)
plt.show()
```



자세히 보면 위아래 두개의 층으로 나누어짐이 보여지지만, pca에 비하면 구분되는 성질이 약하다.

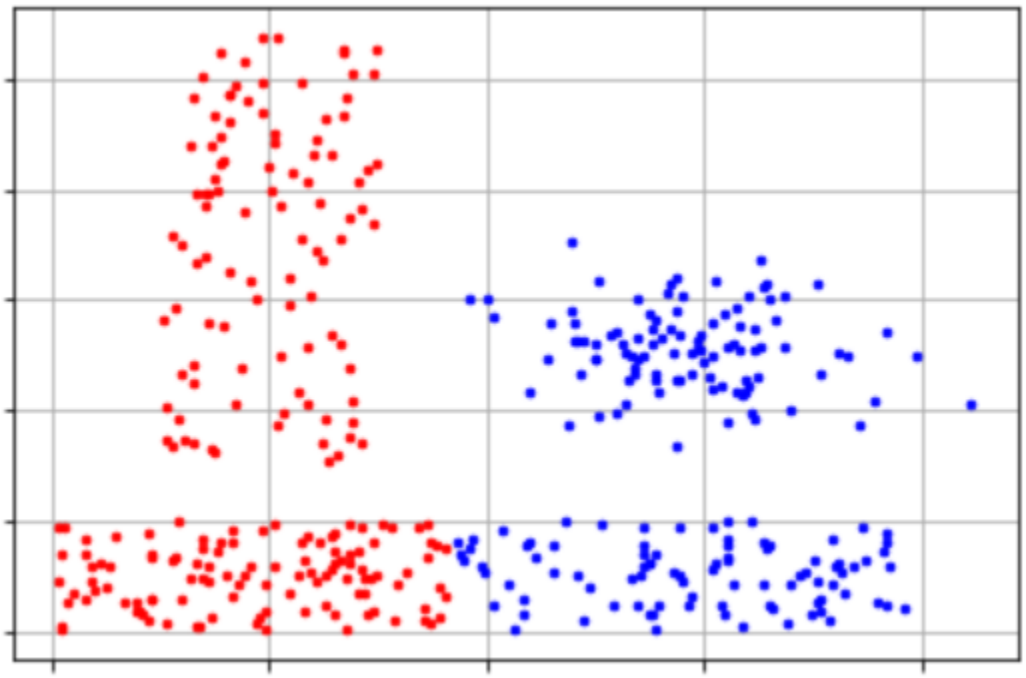
In [91]:
```
#2-1 k-means
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(data)
y_kmeans = kmeans.predict(data)
plt.scatter(data[:, 0], data[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.show()
```



4개의 클러스터를 기준으로 다른 색깔을 적용 하였다. 겹치는 부분이 있긴 하지만 나누어 지기는 했다.

#2-2

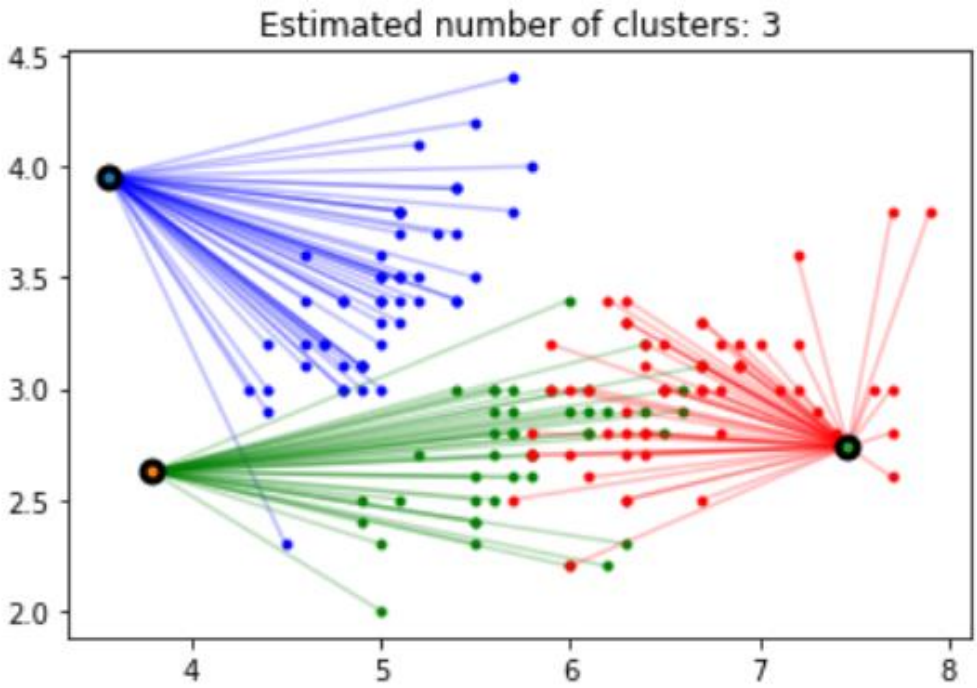
```
from pyclustering.cluster.kmedoids      import kmedoids
from pyclustering.cluster import cluster_visualizer
from sklearn.metrics.pairwise import pairwise_distances
import numpy as np
from numpy import genfromtxt
from pyclustering.utils import read_sample;
from pyclustering.samples.definitions import FCPS_SAMPLES;
data = genfromtxt('data.csv', delimiter=',')
data = data[1:]
data = read_sample(FCPS_SAMPLES.SAMPLE_LSUN);
D = pairwise_distances(data, metric='euclidean')
initial_medoids = [1, 10]
x = kmedoids(data,initial_medoids, 4)
x.process();
clusters = x.get_clusters();
visualizer = cluster_visualizer();
visualizer.append_clusters(clusters, data);
visualizer.show();
```



Pyclustering 모듈을 이용하여 데이터를 4개의 클러스터링을 만들었다. 사실상 가장 정확하게 나누어졌다고 할 수 있다. 최초에는 10개씩 쪼개고, 이를 누적하여서 나타냈다.

```
#2-3 affine propogation
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import AffinityPropagation
from sklearn import metrics
from itertools import cycle
centers = [[4, 3], [5, 2], [7, 3]]
X, labels_true = make_blobs(n_samples=data.size, centers=centers, cluster_std=0.5, random_state=0)
model = AffinityPropagation(preference=-50).fit(data)
cluster_centers_indices = model.cluster_centers_indices_
labels = model.labels_
n_clusters_ = len(cluster_centers_indices)
colors = cycle('bgrcmykbgrcmykbgrcmykbgrcmyk')
for k, col in zip(range(n_clusters_), colors):
    class_members = labels == k
    cluster_center = X[cluster_centers_indices[k]]
    plt.plot(data[class_members, 0], data[class_members, 1], col + '.')
    for x in data[class_members]:
        plt.plot([cluster_center[0], x[0]], [cluster_center[1], x[1]], col, alpha=0.25)
    plt.plot(cluster_center[0], cluster_center[1], 'o', mec='k', mew=3, markersize=7)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```



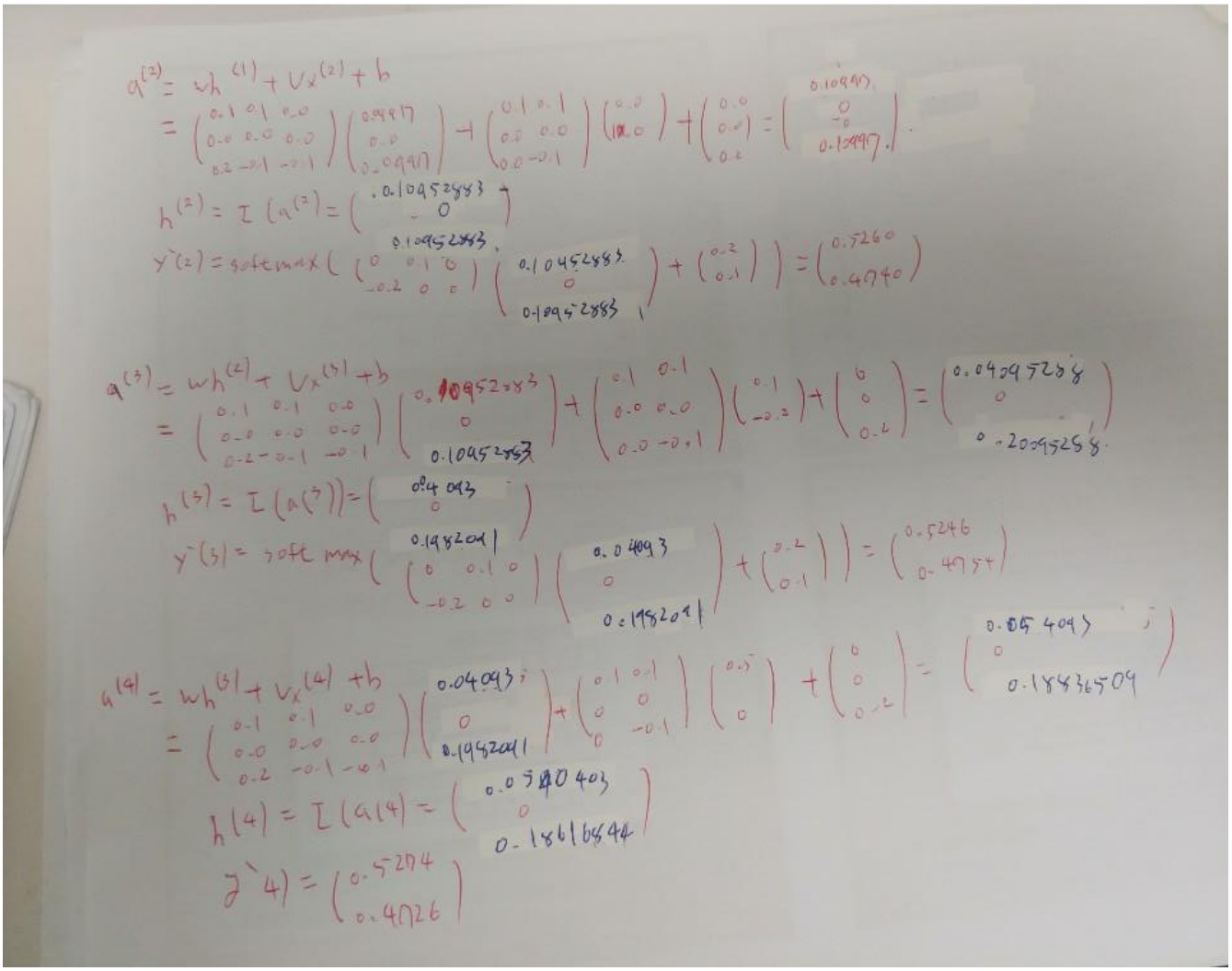
임의의 점을 기준으로 진행을 했다. 역시 겹치는 부분이 존재하지만 초기에 적절한 점을 잘 선택해서 kmeans에 비해 클러스터가 잘 형성 되어 있다.

2-4 비교

k-means 는 k-medoid 에 비해 수행시간이 더 적게 걸리지만, k-medoid에 비해 성능이 떨어지는 경향이 나타난다. 왜냐하면 평균값이기 때문에 잡음에 영향을 많이 받기 때문이다. 그에 반해 k-medoid는 잡음에 덜 영향을 받기 때문에 성능이 더 좋다. 친밀도 전파와 비교했을 때는 초기에 어떤 값을 설정 하느냐에 따라 성능이 달라지는 것 같다. 친밀도 전파는 k-medoid와 비슷한 결과를 보인다. 여기서 k-means의 성능을 높이려면 초기에 랜덤하게 선택이 잘된다면 우수한 성능을

보이게 된다. 여기서는 k-medoid의 성능이 가장 우수하게 나타났지만, 이는 수행을 어떻게 하느냐에 따라서 달라진다. 즉 초기에 어떤 값을 선택하는지에 따라 성능이 달라진다. 결론적으로 k-means의 성능이 가장 떨어지고(대신 속도는 빠르다) 나머지 2개의 성능은 비슷하다.

3.



$a^{(5)} = wh^{(4)} + Ux^{(5)} + b = [\ 0.0254405, 0, 0.1821422\ ]$

$h^{(5)} = \tau(a^{(5)}) = [\ 0.0253985, 0, 0.1802010\ ]$

$\hat{y}^{(5)} = [0.5262, 0.4738]$

$a^{[6]} = wh^{(5)} + Ux^{(6)} + b = [\ 0.0125404, 0, 0.1862785\ ]$

$h^{(6)} = [\ 0.0125305, 0, 0.1841550\ ]$

$\hat{y}^{(6)} = [0.5256, 0.4744]$

4.

~~$\frac{\partial J^{(t)}}{\partial o_2^{(t)}}$~~

$$\frac{\partial J^{(t)}}{\partial o_1^{(t)}} = \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial o_1^{(t)}} = \frac{\partial \left(-1 \cdot 2 \frac{\exp(o_1^{(t)})}{\exp(o_1^{(t)}) + \exp(o_2^{(t)})}\right)}{\partial o_1^{(t)}}$$

$$= \frac{\partial \left(-o_1^{(t)} + 1 \cdot 2 \left(\exp(o_1^{(t)}) + \exp(o_2^{(t)})\right)\right)}{\partial o_1^{(t)}}$$

$$= \frac{\exp(o_1^{(t)})}{\exp(o_1^{(t)}) + \exp(o_2^{(t)})} = \hat{y}_1^{(t)}$$

따라서 우리는

$$\left.\begin{aligned}\frac{\partial J^{(t)}}{\partial v_{12}} &= \left(y_1'^{(t)} - 1\right)h_2^{(t)},\ \mathbf{y}^{(t)} = (1,0)^{\mathrm{T}}\text{일 때}\\ \frac{\partial J^{(t)}}{\partial v_{12}} &= y_1'^{(t)}h_2^{(t)},\ \mathbf{y}^{(t)} = (0,1)^{\mathrm{T}}\text{일 때}\end{aligned}\right\}$$

그림과 같이 2개의 식이 있다. 이를 v12를 vji로 y(t)의 요소 개수를 2개에서 n개인 경우로 일반화 할 수 있다. 즉 2개의 식을 하나로 합치는 것이다. 이는 $\frac{\partial J^{(t)}}{\partial v_{ji}} = \left(y_j'^{(t)} - y_j^{(t)}\right)h_i^{(t)}$ 과 같이 만들어 진다. 이는 단지 한순간에 값임으로 전체 값은 여기에 시그마를 추가하면 된다. 따라서 다음과 같다. $\frac{\partial J}{\partial v_{ji}} = \sum_{t=1}^{T}\left(y_j'^{(t)} - y_j^{(t)}\right)h_i^{(t)}$ 하지만 이는 행렬을 이용하여 표현 하면 더 간단하게 만들 수 있다. 즉 벡터형태로 표현하는 것이다. 따라서 8-19 와 같은 식이 만들어 진다. $\frac{\partial J^{(t)}}{\partial \mathbf{o}^{(t)}} = \mathbf{y}'^{(t)} - \mathbf{y}^{(t)}$