[CSE302] Introduction to Machine Learning

< Assignment 3 >

(Deadline: 2022-05-26)

202123008 Jinmin Kim (김진민)

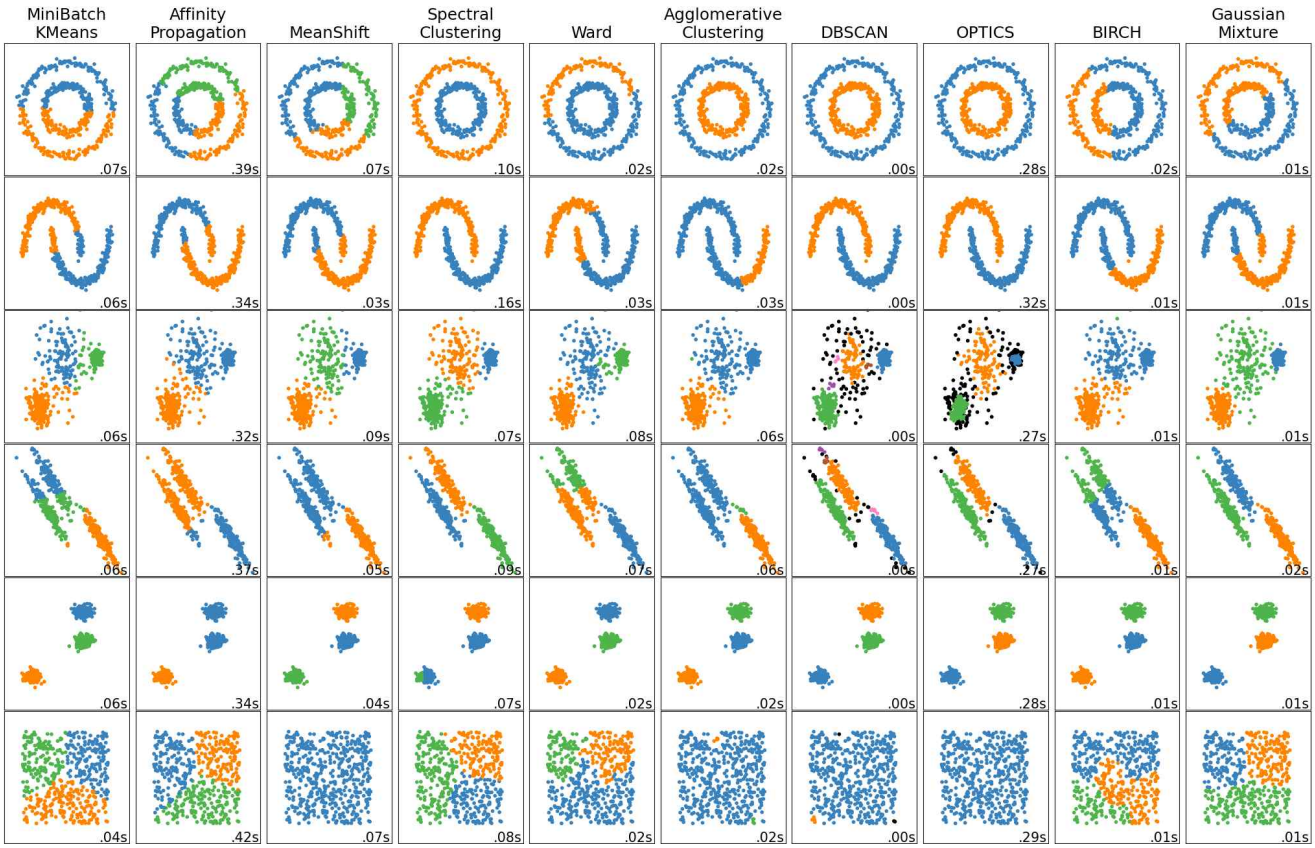
Phone: 010-6266-6099

Mail: rlawlsals@dgist.ac.kr

(1) Run the code below and arrange the results.

*파일: 'hw3_1.py'

** Results



: URL에 있는 코드를 그대로 실행시킨 결과.

** Discussion

- 해당 코드는 6개의 이미지 데이터를 여러 클러스터링 기법을 사용하여 군집화하는 코드이다.

(2) Sample 100 images randomly for each class (total 1000 images) from the MNIST training data set.

*파일: 'hw3_2.py'

** Code

```
19  # MNIST 데이터 불러오기
20  mnist = fetch_openml('mnist_784', version=1)
21  print(mnist.keys())
22  X, y = mnist['data'], mnist['target']
23  X = np.asarray(X)
24  y = np.asarray(y).astype(int)
25
26  print('MNIST data shape: ', X.shape, 'MNIST label shape: ', y.shape)
```

: MNIST 데이터를 불러옴.

```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR', 'details', 'url'])
MNIST data shape:  (70000, 784) MNIST label shape:  (70000,)
```

: MNIST 데이터는 딕셔너리 구조로 되어있음.
데이터 70,000세트가 포함되어 있고, 레이블은 0~9까지의 손글씨 숫자 10종류임.

```
28  # 각 숫자에 해당하는 레이블의 인덱스를 분류
29  for k in range(10):
30      globals()['label{}'.format(k)] = []
31
32  for i in range(len(y)):
33      for k in range(10):
34          if y[i] == k: globals()['label{}'.format(k)].append(i)
35
36  for k in range(10):
37      print('label {} length : '.format(k), len(globals()['label{}'.format(k)]))
38
39  # 각 레이블의 인덱스를 랜덤하게 100개씩 추출
40  for k in range(10):
41      globals()['label{}_100_idx'.format(k)] = random.sample(globals()['label{}'.format(k)], 100)
42      print('sampled label {} length : '.format(k), len(globals()['label{}_100_idx'.format(k)]))
43
44  # 100개씩 추출된 각 레이블 인덱스에 대한 데이터셋을 저장
45  for k in range(10):
46      globals()['data{}_100'.format(k)] = np.zeros(shape=(100, 784))
47      for s in range(100):
48          globals()['data{}_100'.format(k)][s] = X[globals()['label{}_100_idx'.format(k)][s]]
49      print('sampled data {} shape: '.format(k), globals()['data{}_100'.format(k)].shape)
```

1. 각 레이블의 해당하는 데이터들을 추출하기 위해, 레이블의 인덱스를 분류하여 저장함.
2. 추출된 인덱스들 중에서 100개의 인덱스를 랜덤하게 추출함.
3. 추출한 100개의 인덱스에 해당하는 데이터셋을 저장함.

```
label 0 length :  6903
label 1 length :  7877
label 2 length :  6990
label 3 length :  7141
label 4 length :  6824
label 5 length :  6313
label 6 length :  6876
label 7 length :  7293
label 8 length :  6825
label 9 length :  6958
sampled label 0 length :  100
sampled label 1 length :  100
sampled label 2 length :  100
sampled label 3 length :  100
sampled label 4 length :  100
sampled label 5 length :  100
sampled label 6 length :  100
sampled label 7 length :  100
sampled label 8 length :  100
sampled label 9 length :  100
sampled data 0 shape:  (100, 784)
sampled data 1 shape:  (100, 784)
sampled data 2 shape:  (100, 784)
sampled data 3 shape:  (100, 784)
sampled data 4 shape:  (100, 784)
sampled data 5 shape:  (100, 784)
sampled data 6 shape:  (100, 784)
sampled data 7 shape:  (100, 784)
sampled data 8 shape:  (100, 784)
sampled data 9 shape:  (100, 784)
```

: 각 숫자 레이블에 대해 랜덤한 100개의 데이터셋이 추출됨.

```
# 100개씩 추출된 데이터를 합쳐서 1000개의 훈련 데이터 만들기
X_train = np.concatenate([globals()['data{}_100'.format(0)],
                          globals()['data{}_100'.format(1)],
                          globals()['data{}_100'.format(2)],
                          globals()['data{}_100'.format(3)],
                          globals()['data{}_100'.format(4)],
                          globals()['data{}_100'.format(5)],
                          globals()['data{}_100'.format(6)],
                          globals()['data{}_100'.format(7)],
                          globals()['data{}_100'.format(8)],
                          globals()['data{}_100'.format(9)]])
y_train_idx = np.concatenate([globals()['label{}_100_idx'.format(0)],
                              globals()['label{}_100_idx'.format(1)],
                              globals()['label{}_100_idx'.format(2)],
                              globals()['label{}_100_idx'.format(3)],
                              globals()['label{}_100_idx'.format(4)],
                              globals()['label{}_100_idx'.format(5)],
                              globals()['label{}_100_idx'.format(6)],
                              globals()['label{}_100_idx'.format(7)],
                              globals()['label{}_100_idx'.format(8)],
                              globals()['label{}_100_idx'.format(9)]])

print('Final 1000 training data :', X_train.shape)
print('Ground truth label : \n', y[y_train_idx])
```

: 추출된 데이터들을 하나로 합쳐서 훈련 데이터셋과 정답 레이블을 준비함.

```
Final 1000 training data : (1000, 784)
Ground truth label :
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9]
```

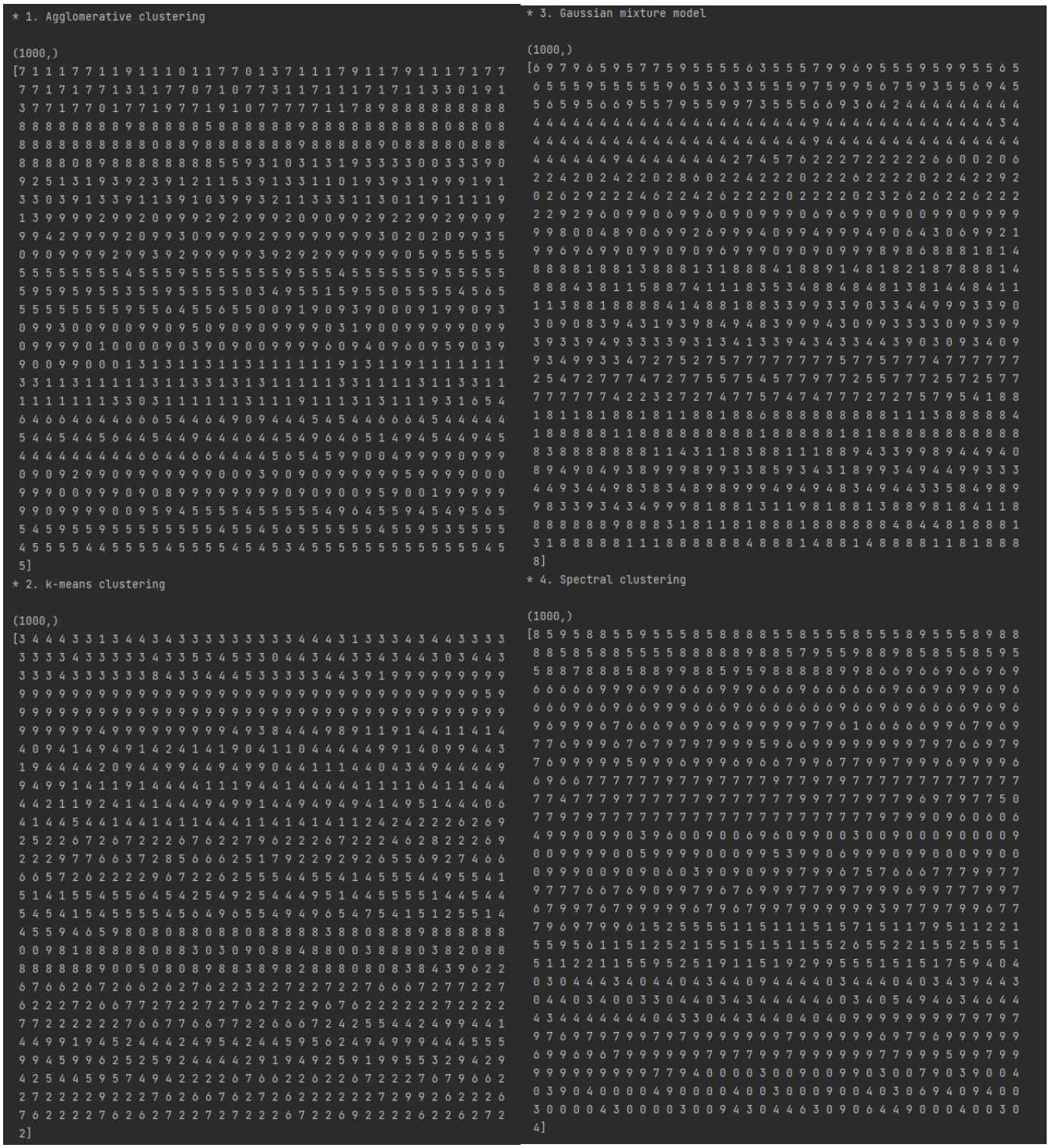
: 최종적으로 (1000, 784)의 크기를 가지는 훈련 데이터셋과 그에 해당하는 정답 레이블을 준비함.

(3) For 1000 images, perform Agglomerative clustering, k-means clustering, Gaussian mixture model, Spectral clustering. (i.e., k = 10)
*파일: 'hw3_2.py'

** Code

```
k = 10

print('* 1. Agglomerative clustering')
print('')
ward = cluster.AgglomerativeClustering(n_clusters=k, linkage="ward")
agglo_labels = ward.fit_predict(X_train)
print(agglo_labels.shape)
print(agglo_labels)

print('* 2. k-means clustering')
print('')
kmeans = cluster.KMeans(n_clusters=k)
kmeans_labels = kmeans.fit_predict(X_train)
print(kmeans_labels.shape)
print(kmeans_labels)

print('* 3. Gaussian mixture model')
print('')
gmm = mixture.GaussianMixture(n_components=k, covariance_type="full")
gmm_labels = gmm.fit_predict(X_train)
print(gmm_labels.shape)
print(gmm_labels)

print('* 4. Spectral clustering')
print('')
spectral = cluster.SpectralClustering(n_clusters=k, eigen_solver="arpack", affinity="nearest_neighbors")
spectral_labels = spectral.fit_predict(X_train)
print(spectral_labels.shape)
print(spectral_labels)
```

: 문제 1에서의 코드를 참고하여 각 클러스터링 모델을 생성하고 훈련 데이터에 대해서 클러스터링을 수행함.



: 각 클러스터링 기법들에 대해 클러스터링된 결과.

클러스터링 기법들은 랜덤하게 초기 중심을 잡고 데이터 구분을 진행하기 때문에, 정답 레이블의 숫자와 매치되지 않을 수 있음.

(4) Based on the clustering results and the labels we know, compute “Rand index” and “mutual information based score”. Explain your findings.

*파일: ‘hw3_2.py’

** Code

```python
ground_truth = y[y_train_idx]
agglomerative_pred = agglo_labels
kmeans_pred = kmeans_labels
gmm_pred = gmm_labels
spectral_pred = spectral_labels

print('')
print('* 1. Rand index')
print('')
rand_score1 = adjusted_rand_score(ground_truth, agglomerative_pred)
rand_score2 = adjusted_rand_score(ground_truth, kmeans_pred)
rand_score3 = adjusted_rand_score(ground_truth, gmm_pred)
rand_score4 = adjusted_rand_score(ground_truth, spectral_pred)
print('Rand score1 :', rand_score1)
print('Rand score2 :', rand_score2)
print('Rand score3 :', rand_score3)
print('Rand score4 :', rand_score4)

print('')
print('* 2. Mutual information based score')
print('')
mutual_score1 = adjusted_mutual_info_score(ground_truth, agglomerative_pred)
mutual_score2 = adjusted_mutual_info_score(ground_truth, kmeans_pred)
mutual_score3 = adjusted_mutual_info_score(ground_truth, gmm_pred)
mutual_score4 = adjusted_mutual_info_score(ground_truth, spectral_pred)
print('Mutual score1 :', mutual_score1)
print('Mutual score2 :', mutual_score2)
print('Mutual score3 :', mutual_score3)
print('Mutual score4 :', mutual_score4)
```

: 각 클러스터링 기법들에 대해서 Rand index 지표와 Mutual information 지표를 계산하였다.

```
* 1. Rand index

Rand score1 : 0.3017694050733901
Rand score2 : 0.25196304632028493
Rand score3 : 0.2759630205408537
Rand score4 : 0.21542596250615473

* 2. Mutual information based score

Mutual score1 : 0.4678288292068427
Mutual score2 : 0.41083793332585833
Mutual score3 : 0.42502973366577007
Mutual score4 : 0.4250204450806622
```

: 실행 결과

** Discussion

- Rand index과 Mutual information은 최적의 성능이면 1에 가까운 값을 나타내고 낮은 성능일 때 0에 가까운 형태를 가진다.
- 우리의 경우 정답 레이블과 클러스터링 레이블의 숫자가 매치되지 않는데, 두 지표는 Incidence Matrix를 구함으로써 다른 클러스터임에도 불구하고 클러스터링 결과의 유사성을 추출해낼 수 있다.
- Rand index는 가능한 모든 쌍의 경우에 대해 정답인 쌍의 개수로 비율을 정의하고, Mutual information은 두 확률변수간의 상호 의존성을 측정한다.

- 우리의 데이터를 두 지표를 이용하여 검증해본 결과, 모두 약 0.2~0.4의 정확도로 측정되었다. 이와 같은 결과가 나온 이유를 추정해보면 다음과 같다.

1. 데이터 전처리 과정에서 정규화나 특징 추출/선택과 같은 처리가 되지 않았기 때문에
2. 클러스터링 파라미터가 적절하게 정의되지 않았기 때문에

참조: https://p829911.github.io/2019/01/05/clustering/

(5) Based on the clustering results, you can get the center of each cluster. Classify the MNIST test data set
using 1−NN classifier and provide accuracy. Explain your findings.
*파일: 'hw3_2.py'

** Code

```
# 테스트 데이터셋 생성 (앞에서부터 20000개의 데이터를 추출)
X_test = np.zeros(shape=(20000, 784))
y_test_idx = np.arange(20000)
for s in range(20000): X_test[s] = X[s]
y_test = y[y_test_idx]
print('test data shape: ', X_test.shape)
print('test label: \n', y_test, '(shape) :', y_test.shape)
```

: 테스트 데이터 20000개를 MNIST 데이터로부터 추출함.

```
test data shape:  (20000, 784)
test label:
 [5 0 4 ... 1 4 2] (shape) : (20000,)
```

: 추출한 결과.

```
# 각 클러스터의 평균을 계산하여 센터를 추출
agglo_centers = np.zeros(shape=(10, 784))
kmeans_centers = np.zeros(shape=(10, 784))
gmm_centers = np.zeros(shape=(10, 784))
spectral_centers = np.zeros(shape=(10, 784))
for k in range(10):
    agglo_avg_temp = np.zeros(shape=(1000, 784))
    kmeans_avg_temp = np.zeros(shape=(1000, 784))
    gmm_avg_temp = np.zeros(shape=(1000, 784))
    spectral_avg_temp = np.zeros(shape=(1000, 784))
    for i in range(1000):
        if agglomerative_pred[i] == k:
            agglo_avg_temp[i] = X_train[i]
        if kmeans_pred[i] == k:
            kmeans_avg_temp[i] = X_train[i]
        if gmm_pred[i] == k:
            gmm_avg_temp[i] = X_train[i]
        if spectral_pred[i] == k:
            spectral_avg_temp[i] = X_train[i]
    agglo_centers[k] = np.mean(agglo_avg_temp, axis=0)
    kmeans_centers[k] = np.mean(kmeans_avg_temp, axis=0)
    gmm_centers[k] = np.mean(gmm_avg_temp, axis=0)
    spectral_centers[k] = np.mean(spectral_avg_temp, axis=0)
```

: 각 클러스터링 기법의 중심을 찾아내기 위해, 0~9까지의 레이블에 대한 평균을 추출함.

```python
# 1-NN classifier 모델을 생성하여 클러스터링 센터 10개에 대한 디시전 바운더리를 생성함
kNN1 = KNeighborsClassifier(n_neighbors=1)
kNN2 = KNeighborsClassifier(n_neighbors=1)
kNN3 = KNeighborsClassifier(n_neighbors=1)
kNN4 = KNeighborsClassifier(n_neighbors=1)
kNN1.fit(agglo_centers, np.arange(10))
kNN2.fit(kmeans_centers, np.arange(10))
kNN3.fit(gmm_centers, np.arange(10))
kNN4.fit(spectral_centers, np.arange(10))

# 학습된 1-NN 모델을 통해 테스트 데이터셋을 예측
predict1 = kNN1.predict(X_test)
predict2 = kNN2.predict(X_test)
predict3 = kNN3.predict(X_test)
predict4 = kNN4.predict(X_test)

# 우리가 아는 레이블과 예측된 값 사이의 정확도 출력
accuracy1 = accuracy_score(y_test, predict1)
accuracy2 = accuracy_score(y_test, predict2)
accuracy3 = accuracy_score(y_test, predict3)
accuracy4 = accuracy_score(y_test, predict4)
print('Accuracy of Agglomerative center using 1-NN : ', accuracy1)
print('Accuracy of Kmeans center using 1-NN : ', accuracy2)
print('Accuracy of GMM center using 1-NN : ', accuracy3)
print('Accuracy of Spectral center using 1-NN : ', accuracy4)
```

: 1-NN classifier 모델을 생성하고, 위에서 구한 클러스터 센터를 학습시킴.
그리고 학습된 1-NN 모델에 대해 정답 레이블과의 정확도를 출력함.

```
Accuracy of Agglomerative center using 1-NN :  0.0226
Accuracy of Kmeans center using 1-NN :  0.0612
Accuracy of GMM center using 1-NN :  0.05895
Accuracy of Spectral center using 1-NN :  0.036
```

: 결과

** Discussion

- 위 결과를 통해 학습이 제대로 진행되지 않았다고 볼 수 있다.

- 이를 개선시켜 학습이 잘 진행되기 위한 방안들은 다음과 같을 것이다.

1. 학습 데이터를 늘린다.
2. 적절한 클러스터링 모델과 파라미터를 정의한다.
3. 커널과 같은 추가적인 기법을 적용한다.