

# [CSE302] Introduction to Machine Learning

## < Assignment 3 >

(Deadline: 2022-05-26)

202123008 Jinmin Kim (김진민)

Phone: 010-6266-6099

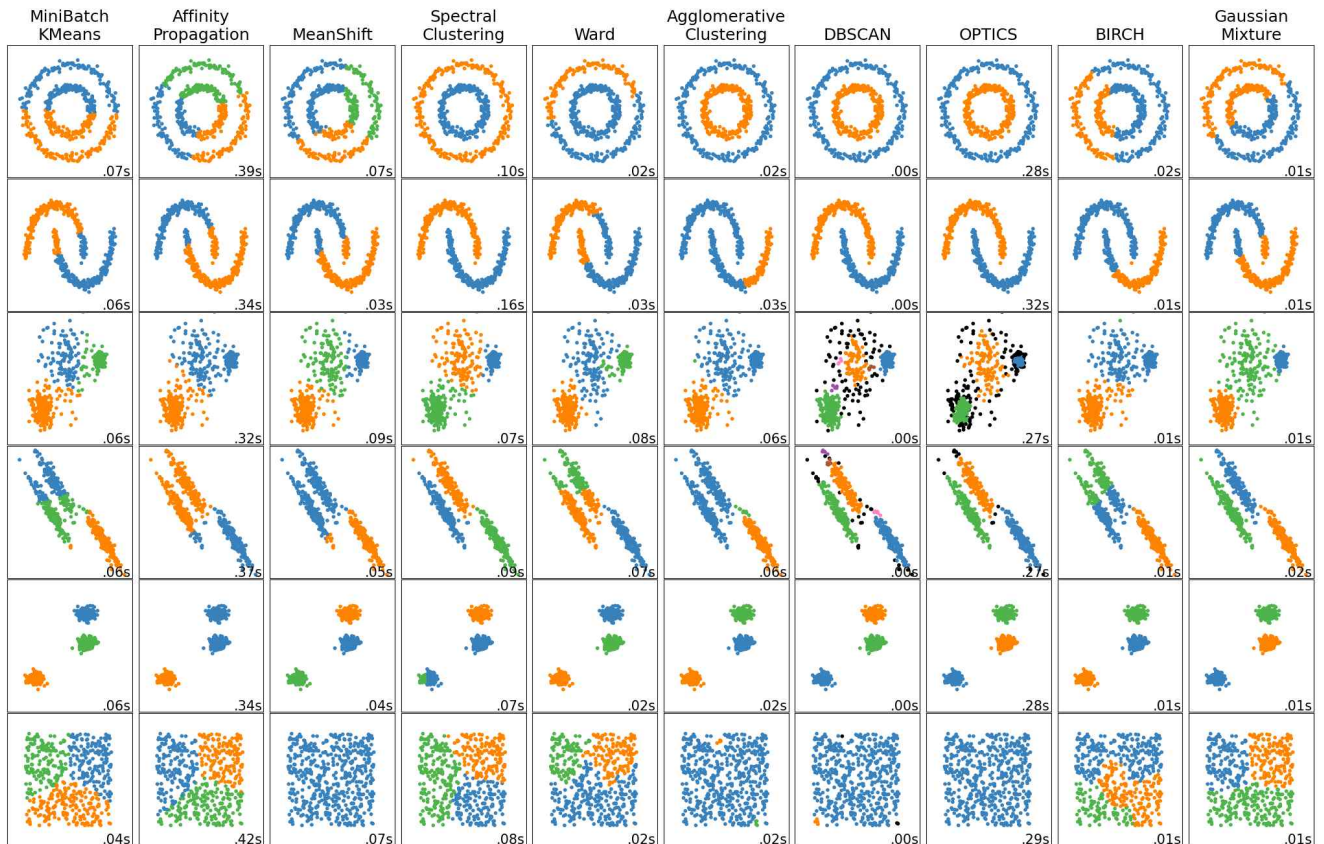
Mail: rlawlsals@dgist.ac.kr



(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.

```

51 # 100개씩 추출된 데이터를 합쳐서 1000개의 훈련 데이터 만들기
52 X_train = np.concatenate([globals()['data{}'.format(i)]_100'.format(0)],
53                             globals()['data{}'.format(i)]_100'.format(1)],
54                             globals()['data{}'.format(i)]_100'.format(2)],
55                             globals()['data{}'.format(i)]_100'.format(3)],
56                             globals()['data{}'.format(i)]_100'.format(4)],
57                             globals()['data{}'.format(i)]_100'.format(5)],
58                             globals()['data{}'.format(i)]_100'.format(6)],
59                             globals()['data{}'.format(i)]_100'.format(7)],
60                             globals()['data{}'.format(i)]_100'.format(8)],
61                             globals()['data{}'.format(i)]_100'.format(9)]])
62 y_train_idx = np.concatenate([globals()['label{}'.format(i)]_100_idx'.format(0)],
63                                globals()['label{}'.format(i)]_100_idx'.format(1)],
64                                globals()['label{}'.format(i)]_100_idx'.format(2)],
65                                globals()['label{}'.format(i)]_100_idx'.format(3)],
66                                globals()['label{}'.format(i)]_100_idx'.format(4)],
67                                globals()['label{}'.format(i)]_100_idx'.format(5)],
68                                globals()['label{}'.format(i)]_100_idx'.format(6)],
69                                globals()['label{}'.format(i)]_100_idx'.format(7)],
70                                globals()['label{}'.format(i)]_100_idx'.format(8)],
71                                globals()['label{}'.format(i)]_100_idx'.format(9)]])
72
73 print('Final 1000 training data :', X_train.shape)
74 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
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 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 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
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
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
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

```
94     k = 10
95
96     print('* 1. Agglomerative clustering')
97     print('')
98     ward = cluster.AgglomerativeClustering(n_clusters=k, linkage="ward")
99     aggro_labels = ward.fit_predict(X_train)
100    print(agglo_labels.shape)
101    print(agglo_labels)
102
103    print('* 2. k-means clustering')
104    print('')
105    kmeans = cluster.KMeans(n_clusters=k)
106    kmeans_labels = kmeans.fit_predict(X_train)
107    print(kmeans_labels.shape)
108    print(kmeans_labels)
109
110    print('* 3. Gaussian mixture model')
111    print('')
112    gmm = mixture.GaussianMixture(n_components=k, covariance_type="full")
113    gmm_labels = gmm.fit_predict(X_train)
114    print(gmm_labels.shape)
115    print(gmm_labels)
116
117    print('* 4. Spectral clustering')
118    print('')
119    spectral = cluster.SpectralClustering(n_clusters=k, eigen_solver="arpack", affinity="nearest_neighbors")
120    spectral_labels = spectral.fit_predict(X_train)
121    print(spectral_labels.shape)
122    print(spectral_labels)
```

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

\* 1. Agglomerative clustering

```
(1000,)  
[7 1 1 1 7 7 1 1 9 1 1 1 0 1 1 7 7 0 1 3 7 1 1 1 7 9 1 1 7 9 1 1 1 7 1 7 7  
7 7 1 7 1 7 7 1 3 1 1 7 7 0 7 1 6 7 7 3 1 1 7 1 1 1 7 1 7 1 1 3 3 0 1 9 1  
3 7 7 1 7 7 0 1 7 7 1 9 7 7 1 9 1 0 7 7 7 7 1 1 7 8 9 8 8 8 8 8 8 8 8 8  
8 8 8 8 8 8 8 8 9 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 8 8 8 8 8 9 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 2 5 1 3 1 9 3 9 2 3 9 1 2 1 1 5 3 9 1 3 3 1 1 0 1 9 3 9 3 1 9 9 9 1 9 1  
3 3 0 3 9 1 3 3 9 1 1 3 9 1 0 3 9 9 3 2 1 1 3 3 3 1 1 3 0 1 1 9 1 1 1 1 9  
1 3 9 9 9 9 2 9 9 2 0 9 9 9 2 9 2 9 9 9 2 0 9 0 9 9 2 9 2 2 9 9 2 9 9 9 9  
9 9 4 2 9 9 9 9 2 0 9 9 3 0 9 9 9 9 2 9 9 9 9 9 9 9 3 0 2 0 2 0 9 9 3 5  
0 9 0 9 9 9 9 2 9 9 3 9 2 9 9 9 9 9 3 9 2 9 2 9 9 9 9 9 0 5 9 5 5 5 5 5  
5 5 5 5 5 5 5 5 4 5 5 5 9 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 9 5 9 5 9 5 5 3 5 5 5 9 5 5 5 5 5 0 3 4 9 5 5 1 5 9 5 5 0 5 5 5 4 5 6 5  
5 5 5 5 5 5 5 5 9 5 5 5 6 4 5 5 5 5 5 0 0 9 1 9 0 9 3 9 0 0 0 9 1 9 9 0 9 3  
0 9 9 3 0 0 9 0 0 9 9 0 9 5 0 9 0 9 0 9 9 9 9 0 3 1 9 0 0 9 9 9 9 9 9 0 9 9  
0 9 9 9 9 0 1 0 0 0 0 9 0 3 9 0 9 0 0 9 9 9 9 6 0 9 4 0 9 6 0 9 5 9 0 3 9  
9 0 0 9 9 0 0 0 1 3 1 3 1 1 3 1 1 3 1 1 1 1 1 1 9 1 3 1 1 9 1 1 1 1 1 1 1  
3 3 1 1 3 1 1 1 1 1 3 1 1 3 1 3 1 3 1 1 1 1 1 1 3 3 1 1 1 1 3 1 1 3 3 1 1  
1 1 1 1 1 1 1 3 3 0 3 1 1 1 1 1 1 3 1 1 1 9 1 1 1 3 1 3 1 1 1 9 3 1 6 5 4  
6 4 6 6 4 6 4 4 6 6 6 6 5 4 4 6 4 9 0 9 4 4 4 5 4 5 4 4 6 6 6 4 5 4 4 4 4 4  
5 4 4 5 4 4 5 6 4 4 5 4 4 9 4 4 4 6 4 4 5 4 9 6 4 6 5 1 4 9 4 5 4 4 9 4 5  
4 4 4 4 4 4 4 4 6 6 4 4 6 6 4 4 4 4 5 6 5 4 5 9 9 0 0 4 9 9 9 9 0 9 9 9  
0 9 0 9 2 9 9 0 9 9 9 9 9 9 9 0 0 9 3 9 0 9 0 9 9 9 9 9 9 5 9 9 9 9 0 0 0  
9 9 9 0 0 9 9 9 0 9 0 8 9 9 9 9 9 9 9 9 0 9 0 9 0 0 9 5 9 0 0 1 9 9 9 9 9  
9 9 0 9 9 9 9 9 0 9 5 9 4 5 5 5 5 4 5 5 5 5 5 4 9 6 4 5 5 9 4 5 4 9 5 6 5  
4 5 9 5 5 5 9 5 5 5 5 5 5 5 4 5 5 4 5 6 5 5 5 5 5 5 4 5 5 9 3 5 5 5 5  
4 5 5 5 5 4 4 5 5 5 5 4 5 5 5 5 4 5 4 5 3 4 5 5 5 5 5 5 5 5 5 5 5 5 4 5  
5]
```

\* 2. k-means clustering

```
(1000,)  
[3 4 4 4 3 3 1 3 4 4 3 4 3 3 3 3 3 3 3 3 4 4 4 3 1 3 3 3 4 3 4 4 3 3 3 3  
3 3 3 3 4 3 3 3 3 3 4 3 3 5 3 4 5 3 3 0 4 4 3 4 4 3 3 4 3 4 3 0 3 3 4 4 3  
3 3 3 4 3 3 3 3 3 3 8 4 3 3 4 4 4 5 3 3 3 3 3 4 4 3 9 1 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 4 9 9 9 9 9 9 9 9 9 4 9 3 8 4 4 4 9 8 9 1 1 9 1 4 4 1 1 4 1 4  
4 0 9 4 1 4 9 4 9 1 4 2 4 1 4 1 9 0 4 1 1 0 4 4 4 4 4 9 9 1 4 0 9 9 4 4 3  
1 9 4 4 4 4 2 0 9 4 4 9 9 4 4 9 4 9 9 0 4 4 1 1 1 4 4 0 4 3 4 9 4 4 4 4 9  
9 4 9 9 1 4 1 1 9 1 4 4 4 4 1 1 1 9 4 4 1 4 4 4 4 4 1 1 1 6 4 1 1 4 4 4  
4 4 2 1 1 9 2 4 1 4 1 4 4 4 9 4 9 1 4 4 9 4 9 4 9 4 1 4 9 5 1 4 4 4 0 6  
4 1 4 4 5 4 4 1 4 4 1 4 1 1 4 4 4 1 1 4 1 4 1 4 1 1 2 4 2 4 2 2 2 6 2 6 9  
2 5 2 2 6 7 2 6 7 2 2 2 6 7 6 2 2 7 9 6 2 2 2 6 7 2 2 2 4 6 2 8 2 2 2 6 6  
2 2 2 9 7 7 6 6 3 7 2 8 5 6 6 6 2 5 1 7 9 2 2 9 2 9 2 6 5 5 6 9 2 7 4 6 6  
6 6 5 7 2 6 2 2 2 9 6 7 2 2 6 2 5 5 4 4 5 5 4 1 4 5 5 4 5 4 9 5 4 1  
5 1 4 1 5 5 4 5 5 6 4 5 4 2 5 4 9 2 5 4 4 4 9 5 1 4 4 5 5 5 5 1 4 4 5 4 4  
5 4 5 4 1 5 4 5 5 5 5 4 5 6 4 9 6 5 5 4 9 4 9 6 5 4 7 5 4 1 5 1 2 5 5 1 4  
4 5 5 9 4 6 5 9 8 0 8 0 8 8 0 8 8 0 8 8 8 8 8 8 3 8 8 0 8 8 8 9 8 8 8 8 8 8  
0 0 9 8 1 8 8 8 8 8 8 8 8 3 0 3 0 9 0 8 8 4 8 8 8 0 3 8 8 8 0 3 8 2 0 8 8  
8 8 8 8 8 8 9 0 0 5 0 8 0 8 9 8 8 3 8 9 8 2 8 8 8 0 8 0 8 3 8 4 3 9 6 2 2  
6 7 6 6 2 6 7 2 6 6 2 6 2 7 6 2 2 3 2 2 7 2 2 7 2 2 7 6 6 6 7 2 7 7 2 2 7  
6 2 2 2 7 2 6 6 7 7 2 7 2 2 7 2 7 6 2 7 2 2 9 6 7 6 2 2 2 2 2 2 7 2 2 2 2  
7 7 2 2 2 2 2 2 7 6 6 7 7 6 6 7 7 2 2 6 6 6 7 2 4 2 5 5 4 4 2 4 9 9 4 4 1  
4 4 9 9 1 9 4 5 2 4 4 4 2 4 9 5 4 2 4 4 5 9 5 6 2 4 9 4 9 9 4 4 4 5 5 5  
9 9 4 5 9 9 6 2 5 2 5 9 2 4 4 4 4 2 9 1 9 4 9 2 5 9 1 9 9 5 5 3 2 9 4 2 9  
4 2 5 4 4 5 9 5 7 4 9 4 2 2 2 2 6 7 6 6 2 2 6 2 2 6 7 2 2 2 7 6 7 9 6 6 2  
2 7 2 2 2 2 9 2 2 2 7 6 2 6 6 7 6 2 7 2 6 2 2 2 2 2 2 7 2 9 9 2 6 2 2 2 6  
7 6 2 2 2 2 7 6 2 6 2 7 2 2 7 2 7 2 2 2 6 7 2 2 6 9 2 2 2 2 6 2 2 6 2 7 2  
2]
```

\* 3. Gaussian mixture model

```
(1000,)  
[6 9 7 9 6 5 9 5 7 7 5 9 5 5 5 5 6 3 5 5 5 7 9 9 6 9 5 5 5 9 5 9 9 5 5 6 5  
6 5 5 5 9 5 5 5 5 5 9 6 5 3 6 3 3 5 5 5 9 7 5 9 9 5 6 7 5 9 3 5 5 6 9 4 5  
5 6 5 9 5 6 6 9 5 5 7 9 5 5 9 9 7 3 5 5 5 6 6 9 3 6 4 2 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 9 4 4 4 4 4 4 4 4 4 4 4 3 4  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 9 4 4 4 4 4 4 4 4 4 4 4 4 4  
4 4 4 4 4 4 9 4 4 4 4 4 4 2 7 4 5 7 6 2 2 2 7 2 2 2 2 2 2 6 6 0 0 2 0 6  
2 2 4 2 0 2 4 2 2 0 2 8 6 0 2 2 4 2 2 2 0 2 2 2 6 2 2 2 2 0 2 2 4 2 2 9 2  
0 2 6 2 9 2 2 2 4 6 2 2 4 2 6 2 2 2 2 0 2 2 2 2 0 2 3 2 6 2 6 2 2 6 2 2 2  
2 2 9 2 9 6 0 9 9 0 6 9 9 6 0 9 0 9 9 9 9 0 6 9 6 9 9 0 9 0 9 9 0 9 9 9 9  
9 9 8 0 0 4 8 9 0 6 9 9 2 6 9 9 9 4 0 9 9 4 9 9 9 4 9 0 6 4 3 0 6 9 9 2 1  
9 9 6 9 6 9 9 0 9 9 0 9 0 9 6 9 9 9 0 9 0 9 0 9 9 9 8 9 8 6 8 8 8 1 8 1 4  
8 8 8 8 1 8 8 1 3 8 8 8 1 3 1 8 8 8 4 1 8 8 9 1 4 8 1 8 2 1 8 7 8 8 8 1 4  
8 8 8 4 3 8 1 1 5 8 8 7 4 1 1 1 8 3 5 3 4 8 8 4 8 4 8 1 3 8 1 4 4 8 4 1 1  
1 1 3 8 8 1 8 8 8 8 4 1 4 8 8 1 8 8 3 3 9 9 3 3 9 0 3 3 4 4 9 9 9 3 3 9 0  
3 0 9 0 8 3 9 4 3 1 9 3 9 8 4 9 4 8 3 9 9 9 4 3 0 9 9 3 3 3 3 0 9 9 3 9 9  
9 3 3 3 9 4 9 3 3 3 3 9 3 1 3 4 1 3 3 9 4 3 4 3 3 4 3 9 0 3 0 9 3 4 0 9  
3 3 4 9 9 3 3 4 7 2 7 5 2 7 5 7 7 7 7 7 7 7 5 7 7 5 7 7 7 4 7 7 7 7 7 7  
2 5 4 7 2 7 7 7 4 7 2 7 7 5 7 5 4 5 7 7 9 7 7 2 5 7 7 7 2 5 7 2 5 7 7  
7 7 7 7 7 7 4 2 2 3 2 7 2 7 4 7 7 5 7 4 7 4 7 7 7 2 7 2 7 5 7 9 5 4 1 8 8  
1 8 1 1 8 1 8 8 1 8 1 1 8 8 1 8 8 6 8 8 8 8 8 8 8 8 8 1 1 1 3 8 8 8 8 8 4  
1 8 8 8 8 8 1 1 8 8 8 8 8 8 8 8 1 8 8 8 8 8 1 8 1 8 8 8 8 8 8 8 8 8 8 8  
8 3 8 8 8 8 8 8 8 1 1 4 3 1 1 8 3 8 8 1 1 1 8 8 9 4 3 3 9 9 8 9 4 4 9 4 0  
8 9 4 9 0 4 9 3 8 9 9 9 8 9 9 3 3 8 5 9 3 4 3 1 8 9 9 3 4 9 4 4 9 9 3 3 3  
4 4 9 3 4 4 9 8 3 8 3 4 8 9 8 9 9 9 4 9 4 9 4 8 3 4 9 4 4 3 3 5 8 4 9 8 9  
9 8 3 3 9 3 4 3 4 9 9 9 8 1 8 8 1 3 1 1 9 8 1 8 8 1 3 8 8 9 8 1 8 4 1 1 8  
8 8 8 8 8 8 9 8 8 8 3 1 8 1 1 8 1 8 8 8 1 8 8 8 8 8 4 8 4 8 1 8 8 8 1  
3 1 8 8 8 8 8 1 1 1 8 8 8 8 8 8 4 8 8 8 1 4 8 8 8 1 1 8 1 8 8 8  
8]
```

\* 4. Spectral clustering

```
(1000,)  
[8 5 9 5 8 8 5 5 9 5 5 5 8 5 8 8 8 8 8 5 5 8 5 5 5 8 5 5 5 8 9 5 5 5 8 9 8 8  
8 8 5 8 8 8 5 5 5 5 8 8 8 8 8 8 8 8 5 7 9 5 5 9 8 8 9 8 5 8 5 5 8 5 9 5  
5 8 8 7 8 8 8 5 8 8 9 9 8 8 5 9 5 9 8 8 8 8 8 9 9 8 6 6 9 6 6 9 6 6 9  
6 6 6 6 6 9 9 9 6 9 9 6 6 6 9 9 9 6 6 6 6 6 6 6 6 6 6 6 6 9 9 6 9 6 6  
6 6 6 9 6 6 9 6 6 9 9 9 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 9 6 6 6 9 6  
9 6 9 9 9 6 7 6 6 6 9 6 9 6 9 6 9 9 9 9 7 9 6 1 6 6 6 6 6 9 9 6 7 9 6 9  
7 7 6 9 9 9 6 7 6 7 9 7 9 7 9 9 9 5 9 6 6 9 9 9 9 9 9 9 7 9 7 6 6 9 7 9  
7 6 9 9 9 9 9 5 9 9 9 6 9 9 9 6 9 6 6 7 9 9 6 7 7 9 9 7 9 9 6 9 9 9 6  
6 9 6 6 7 7 7 7 7 9 7 7 9 7 7 7 7 7 9 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
7 7 4 7 7 7 9 7 7 7 7 7 9 7 7 7 7 7 7 9 9 7 7 7 9 7 7 9 6 9 7 9 7 7 5 0  
7 7 9 7 9 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 9 9 9 0 6 0 6 0 6  
4 9 9 9 9 9 0 3 9 6 0 0 9 0 0 6 9 6 0 9 9 0 0 3 0 0 9 0 0 9 0 0 0 0 9  
0 0 9 9 9 9 0 0 5 9 9 9 9 0 0 0 9 9 5 3 9 9 0 6 9 9 9 9 0 0 9 0 0 9 9 0  
0 9 9 9 0 0 9 0 9 0 6 0 3 9 0 9 0 9 9 9 7 9 9 6 7 5 7 6 6 6 7 7 7 9 9 7 7  
9 7 7 7 6 6 7 6 9 0 9 9 7 9 6 7 6 9 9 9 7 7 9 9 7 9 9 9 6 9 9 7 7 9 9 7  
6 7 9 9 7 6 7 9 9 9 9 6 7 9 6 7 9 9 7 9 9 9 9 9 3 9 7 7 9 7 9 9 6 7 7  
7 9 6 9 7 9 9 6 1 5 2 5 5 5 1 1 5 1 1 1 5 1 5 7 1 5 1 1 7 9 5 1 1 2 2 1  
5 5 9 5 6 1 1 5 1 2 5 2 1 5 5 1 5 1 5 1 1 5 5 2 6 5 5 2 2 1 5 5 2 5 5 1  
5 1 1 2 2 1 1 5 5 9 5 2 5 1 9 1 1 5 1 9 2 9 9 5 5 5 1 5 1 5 1 7 5 9 4 0 4  
0 3 0 4 4 3 4 0 4 4 0 4 3 4 4 0 9 4 4 4 4 0 3 4 4 4 0 4 0 3 4 3 9 4 4 3  
0 4 4 0 3 4 0 0 3 3 0 4 4 0 3 4 3 4 4 4 4 6 0 3 4 0 5 4 9 4 6 3 4 6 4 4  
4 3 4 4 4 4 4 4 4 0 4 3 3 0 4 4 3 4 4 0 4 0 4 0 9 9 9 9 9 9 9 9 7 9 7 9 7  
9 7 6 9 7 9 7 9 7 9 7 9 9 9 9 9 9 7 9 9 9 9 9 6 9 7 9 6 9 9 9 9 9 9  
6 9 9 6 9 6 7 9 9 9 9 9 9 7 9 7 7 9 9 9 9 9 9 7 7 9 9 9 5 9 9 7 9 9  
9 9 9 9 9 9 9 9 9 7 7 9 4 0 0 0 0 3 0 0 9 0 0 9 0 3 0 0 7 9 0 3 9 0 0 4  
0 3 9 0 4 0 0 0 0 4 9 0 0 0 0 4 0 0 3 0 0 0 9 0 0 4 0 3 0 6 9 4 0 9 4 0 0  
3 0 0 0 0 4 3 0 0 0 0 3 0 0 9 4 3 0 4 4 6 3 0 9 0 6 4 4 9 0 0 0 4 0 0 3 0  
4]
```

: 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

```
128 ground_truth = y[y_train_idx]
129 agglomerative_pred = agglo_labels
130 kmeans_pred = kmeans_labels
131 gmm_pred = gmm_labels
132 spectral_pred = spectral_labels
133
134 print('')
135 print('* 1. Rand index')
136 print('')
137 rand_score1 = adjusted_rand_score(ground_truth, agglomerative_pred)
138 rand_score2 = adjusted_rand_score(ground_truth, kmeans_pred)
139 rand_score3 = adjusted_rand_score(ground_truth, gmm_pred)
140 rand_score4 = adjusted_rand_score(ground_truth, spectral_pred)
141 print('Rand score1 :', rand_score1)
142 print('Rand score2 :', rand_score2)
143 print('Rand score3 :', rand_score3)
144 print('Rand score4 :', rand_score4)
145
146 print('')
147 print('* 2. Mutual information based score')
148 print('')
149 mutual_score1 = adjusted_mutual_info_score(ground_truth, agglomerative_pred)
150 mutual_score2 = adjusted_mutual_info_score(ground_truth, kmeans_pred)
151 mutual_score3 = adjusted_mutual_info_score(ground_truth, gmm_pred)
152 mutual_score4 = adjusted_mutual_info_score(ground_truth, spectral_pred)
153 print('Mutual score1 :', mutual_score1)
154 print('Mutual score2 :', mutual_score2)
155 print('Mutual score3 :', mutual_score3)
156 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

```
162 # 테스트 데이터셋 생성 (앞에서부터 20000개의 데이터를 추출)
163 X_test = np.zeros(shape=(20000, 784))
164 y_test_idx = np.arange(20000)
165 for s in range(20000): X_test[s] = X[s]
166 y_test = y[y_test_idx]
167 print('test data shape: ', X_test.shape)
168 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.

```
170 # 각 클러스터의 평균을 계산하여 센터를 추출
171 agglo_centers = np.zeros(shape=(10, 784))
172 kmeans_centers = np.zeros(shape=(10, 784))
173 gmm_centers = np.zeros(shape=(10, 784))
174 spectral_centers = np.zeros(shape=(10, 784))
175 for k in range(10):
176     agglo_avg_temp = np.zeros(shape=(1000, 784))
177     kmeans_avg_temp = np.zeros(shape=(1000, 784))
178     gmm_avg_temp = np.zeros(shape=(1000, 784))
179     spectral_avg_temp = np.zeros(shape=(1000, 784))
180     for i in range(1000):
181         if agglomerative_pred[i] == k:
182             agglo_avg_temp[i] = X_train[i]
183         if kmeans_pred[i] == k:
184             kmeans_avg_temp[i] = X_train[i]
185         if gmm_pred[i] == k:
186             gmm_avg_temp[i] = X_train[i]
187         if spectral_pred[i] == k:
188             spectral_avg_temp[i] = X_train[i]
189     agglo_centers[k] = np.mean(agglo_avg_temp, axis=0)
190     kmeans_centers[k] = np.mean(kmeans_avg_temp, axis=0)
191     gmm_centers[k] = np.mean(gmm_avg_temp, axis=0)
192     spectral_centers[k] = np.mean(spectral_avg_temp, axis=0)
```

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

```

194 # 1-NN classifier 모델을 생성하여 클러스터링 센터 10개에 대한 디스턴스 바운더리를 생성함
195 kNN1 = KNeighborsClassifier(n_neighbors=1)
196 kNN2 = KNeighborsClassifier(n_neighbors=1)
197 kNN3 = KNeighborsClassifier(n_neighbors=1)
198 kNN4 = KNeighborsClassifier(n_neighbors=1)
199 kNN1.fit(agglo_centers, np.arange(10))
200 kNN2.fit(kmeans_centers, np.arange(10))
201 kNN3.fit(gmm_centers, np.arange(10))
202 kNN4.fit(spectral_centers, np.arange(10))
203
204 # 학습된 1-NN 모델을 통해 테스트 데이터셋을 예측
205 predict1 = kNN1.predict(X_test)
206 predict2 = kNN2.predict(X_test)
207 predict3 = kNN3.predict(X_test)
208 predict4 = kNN4.predict(X_test)
209
210 # 우리가 아는 레이블과 예측된 값 사이의 정확도 출력
211 accuracy1 = accuracy_score(y_test, predict1)
212 accuracy2 = accuracy_score(y_test, predict2)
213 accuracy3 = accuracy_score(y_test, predict3)
214 accuracy4 = accuracy_score(y_test, predict4)
215 print('Accuracy of Agglomerative center using 1-NN : ', accuracy1)
216 print('Accuracy of Kmeans center using 1-NN : ', accuracy2)
217 print('Accuracy of GMM center using 1-NN : ', accuracy3)
218 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.