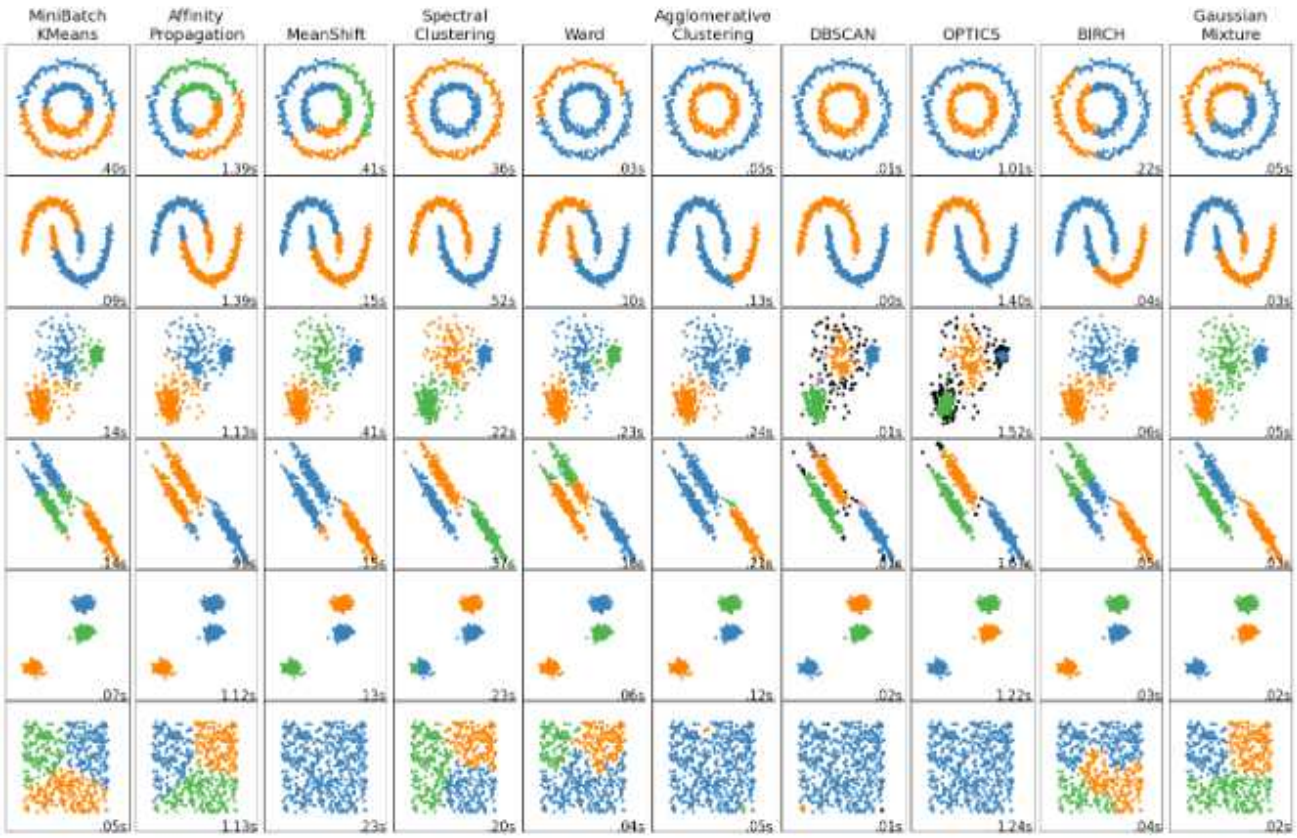
과제명	HW3	학번	201611182	이름	최동현

보고서 내용	주요코드, 결과, 결과에 대한 분석을 기술한 보고서(jupyter notebook 활용)

문제 번호	내용
1번	○ 코드 실행 결과  -코드 실행 결과, 여러 clustering에 대해서 어떻게 cluster되었는지 정렬되었다. -다양한 데이터에서 Spectral Clustering이 가장 잘 분류하는 것을 확인할 수 있었다.

2번	○ Mnist data set 불러오기
	`import tensorflow as tf`
	`from tensorflow import keras`
	`from tensorflow.keras import layers`
	`import random`
	`import numpy as np`
	
	`mnist = tf.keras.datasets.mnist`
	
	`(X_train, y_train), (X_test, y_test) = mnist.load_data()`
	`X_train, X_test = X_train / 255.0, X_test / 255.0`
	`print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)`
	
	`nsamples, nx, ny = X_train.shape`
	`X_train = X_train.reshape((nsamples,nx*ny))`
	`nsamples, nx, ny = X_test.shape`
	`X_test = X_test.reshape((nsamples,nx*ny))`
	`print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)`
	
	`(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)`
	`(60000, 784) (60000,) (10000, 784) (10000,)`
	
	-다음과 같이 Mnist data를 불러왔다.

○ class별로 100개의 이미지씩 총 1000개의 이미지 랜덤 추출 코드

```
random.seed(100)
X_list = [[],[],[],[],[],[],[],[],[],[]]

for i in range(len(X_train)):
    if y_train[i] == 0:
        X_list[0].append(X_train[i])
    elif y_train[i] == 1:
        X_list[1].append(X_train[i])
    elif y_train[i] == 2:
        X_list[2].append(X_train[i])
    elif y_train[i] == 3:
        X_list[3].append(X_train[i])
    elif y_train[i] == 4:
        X_list[4].append(X_train[i])
    elif y_train[i] == 5:
        X_list[5].append(X_train[i])
    elif y_train[i] == 6:
        X_list[6].append(X_train[i])
    elif y_train[i] == 7:
        X_list[7].append(X_train[i])
    elif y_train[i] == 8:
        X_list[8].append(X_train[i])
    elif y_train[i] == 9:
        X_list[9].append(X_train[i])

#X
for i in range(10):
    X_list[i] = random.sample(X_list[i], 100)
print(len(X_list[4]))

new_X_train = np.concatenate(X_list, axis=0)
print(new_X_train.shape)

#y
y_list = []
for i in range(10):
    y_list.append([i]*100)

new_y_train = np.concatenate(y_list, axis=0)
print(len(new_y_train))
print(new_y_train.shape)
```

```
100
(1000, 784)
1000
(1000,)
```

-X_list의 빈 리스트를 만들어 주어서 class별로 나눠서 전부 데이터들을 배분했다.
-배분된 X_list에 대하여 랜덤으로 100개씩 뽑아서 new_X_train에 저장하였다.
-최종적으로 shape해봤을 때, 1000개가 랜덤 추출됨을 확인할 수 있었다.

3번

○ 각 클러스터에 fitting

```
from sklearn.cluster import AgglomerativeClustering

agg = AgglomerativeClustering(n_clusters=10).fit_predict(new_X_train)
```

```
from sklearn.cluster import KMeans

Kmeans = KMeans(n_clusters=10).fit_predict(new_X_train)
```

```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=10, random_state=0).fit_predict(new_X_train)
```

```
from sklearn.cluster import SpectralClustering

sc = SpectralClustering(n_clusters=10).fit_predict(new_X_train)
```

- 랜덤 추출한 1000개의 데이터에 대하여 각 클러스터에 fitting하였다.(k=10)

4번

○ Rand_index 결과

```
#4번
from sklearn.metrics.cluster import rand_score

print("Agglomerative clustering rand_score:", rand_score(new_y_train, agg))
print("k-means clustering rand_score:", rand_score(new_y_train, Kmeans))
print("Gaussian mixture model rand_score:", rand_score(new_y_train, gmm))
print("Spectral clustering rand_score:", rand_score(new_y_train, sc))
```

```
Agglomerative clustering rand_score: 0.9058998998998999
k-means clustering rand_score: 0.8916036036036036
Gaussian mixture model rand_score: 0.8788068068068068
Spectral clustering rand_score: 0.14221821821821823
```

-Rand index는 분류모형의 성능평가 지표인 정분류율(accuracy)과 비슷하다. CLustering 알고리즘의 성능 평가 측도로 사용하여 4개의 clustering에 대해 성능을 평가한 결과, Spectral clustering가 현저히 낮게 나옴을 확인할 수 있었다.

-병합 군집(agglomerative clustering) 알고리즘은 시작할 때 각 포인트를 하나의 클러스터로 지정하고, 그 다음 종료 조건을 만족할 때까지 가장 비슷한 두 클러스터를 합친다.

- KMeans는 원형의 범위에서 군집화를 수행하기 때문에 dataset이 원형의 범위를 가질수록 군집화 효율은 더욱 높아진다.

-Gaussian Mixture Model (GMM)은 이름 그대로 Gaussian 분포가 여러 개 혼합된 clustering 알고리즘이다.

-Spectral clustering은 그래프 기반 클러스터링이다. 그래프 기반이라는 것은 데이터들 간의 상대적인 관계나 연결을 가장 중요한 정보로 활용하였다는 것이다. KMeans 클러스터링은 유클리디언 공간 위의 데이터의 값들 자체의 거리를 기반으로, 군집의 중심을 정하고 그 중심에 가깝게 데이터들을 배정해가는 방식으로 군집을 찾아낸다. 반면에, 그래프 기반으로 접근하면 데이터들의 절대적 위치가 중요한 것이 아니라 그들이 연결되어 있는지를 보고 같은 군집인지 아닌지를 판단하게 된다. 랜덤으로 추출해낸 1,000개의 Mnist data의 경우, 유클리디언 공간 위에 놓고 보면 비교적 군집화가 잘 이루어진다. 하지만 연결성을 본다면 숫자들이 이미지의 중심을 기준으로 형성되어 있기 때문에 Spectral clustering이 Mnist data를 같은 군집으로 파악하는 것이다. 이 때문에 Spectral clustering의 rand_score가 매우 낮음을 확인할 수 있었다.

○ mutual information based score 결과

```
from sklearn.metrics.cluster import mutual_info_score

print("Agglomerative clustering mutual_info_score:", mutual_info_score(new_y_train, agg))
print("k-means clustering mutual_info_score:", mutual_info_score(new_y_train, Kmeans))
print("Gaussian mixture model mutual_info_score:", mutual_info_score(new_y_train, gmm))
print("Spectral clustering mutual_info_score:", mutual_info_score(new_y_train, sc))
```

```
Agglomerative clustering mutual_info_score: 1.4080261759251629
k-means clustering mutual_info_score: 1.2486572156857298
Gaussian mixture model mutual_info_score: 1.1062010639164406
Spectral clustering mutual_info_score: 0.05234772010151926
```

- 사이킷런 패키지의 metrics 서브패키지는 이산확률변수의 상호정보량을 구하는 mutual_info_score 명령을 제공한다.

- mutual_info_score 명령은 각 데이터에 대해서 X,Y 카테고리값을 표시한 2차원 배열을 입력해야 한다.

- mutual_info_score 명령은 상호정보량을 구하는 것이기 때문에 Rand_score가 높았던 Agglomerative clustering, k-means clustering, Gaussian mixture model에 대해서 1보다 큰 값이 나왔다. 상호정보량을 구하는 과정에서 정규화를 해주어서 비교한다면, 1보다 낮은 값으로 직관적으로 확인할 수 있다.

- Spectral clustering이 0.0523 수준의 mutual_info_score가 나왔는데, cluster된 data들이 비교적 같은 군집으로 이루어져 있다. 따라서 종속성이 강하고, 이에 따라 이산확률 값이 낮게 나온 것이다.

5번

○ k-means clustering의 Cluster Class의 Center 찾기

```
Kmeans = KMeans(n_clusters=10).fit(new_X_train)
kmeans_centers = Kmeans.cluster_centers_
kmeans_centers.shape
```

```
(10, 784)
```

- Kmeans에 대해서는 Centers_함수 모듈을 이용하여 의해 쉽게 찾을 수 있었다.

```
from collections import Counter

y_means = []

for i in range(10):
    cnt = Counter(y_list[i])
    y_means.append(cnt.most_common(1)[0][0])

print(y_means)
```

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

- y_train 또한 center shape에 맞게 최빈 함수를 이용하여 10개로 만들었다.

○ k-means clustering에 test data set을 적용하여 predict한 결과

```
k = Kmeans.reshape(10,-1)
Kmeans_X_means = []

for i in range(10):
    cnt = Counter(k[i])
    Kmeans_X_means.append([cnt.most_common(1)[0][0]])

print(Kmeans_X_means)
#print(k)
#print(new_y_train)


from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

#agg_X_means.reshape(1, -1)
knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(kmeans_centers, y_means)
knn_test = knn.predict(X_test)
#print(knn_test.shape)
score = classification_report(knn_test, y_test)
print(score)
```

```
[[9], [8], [0], [5], [7], [5], [1], [2], [6], [7]]
              precision    recall  f1-score   support

           0       0.04      0.08      0.05       512
           1       0.00      0.00      0.00      1010
           2       0.02      0.01      0.01      1199
           3       0.01      0.01      0.01      1619
           4       0.00      0.00      0.00      1300
           5       0.00      0.00      0.00       801
           6       0.01      0.01      0.01       788
           7       0.00      0.00      0.00       732
           8       0.12      0.13      0.12       955
           9       0.02      0.01      0.02      1084

    accuracy                           0.02     10000
   macro avg       0.02      0.03      0.02     10000
weighted avg       0.02      0.02      0.02     10000
```

-kmeans_centers를 1-nn에 fit하고, (10000, 784)의 shape을 가지는 test data set을 prediction하였다. 그 결과를 test data set label(이미지의 answer)와 비교한 결과, accuracy는 0.02였다.

-Mnist 손글씨 이미지를 직관적으로 봐도 각 이미지들에서 숫자가 적힌 위치가 많이 차이가 난다. 그래서 군집화 된 데이터의 center점 즉, 평균을 구하면, 숫자들이 겹쳐져 구분하기 힘들게 된다. 나아가 1-nn으로 가장 가까운 군집에 classifier했지만 대부분 다른 숫자들에 near하여 구분되었다.

-더 좋은 결과를 내기 위한 방법 고안.

1. 이미지의 해상도를 높이는 것이다. Mnist는 28*28로 이루어진 손글씨 이미지이기 때문에 쉽게 군집화하기 어렵다.
2. Mnist 손글씨 이미지 데이터를 전처리 하는 것이다. 숫자 0을 28*28 matrix로 나타낼 때, 진한부분(예를 들면 0~255에서 200이상)만 추출하여 학습데이터로 활용하면 좀 더 명확한 구분이 일어날 수 있을 것 같다.
3. Kmeans가 아닌 그래프 기반의 Clustering 기법 사용
4. graph cut처럼 사용자가 forward ground와 backward ground를 설정하는 것.

○ 다른 Clustering Class의 Center를 찾지 못한 이유 분석

-Agglomerative clustering, Gaussian mixture model, Spectral clustering은 Centers_함수 모듈을 지원하지 않았다. 그렇기 때문에 직접 center를 생성하는 코드를 짜야 했다.

```
a = agg.reshape(10,-1)
agg_X_means = []

for i in range(10):
    cnt = Counter(a[i])
    agg_X_means.append(cnt.most_common(1)[0][0])

agg_X_means = np.array(agg_X_means)
print(agg_X_means)
print(agg_X_means.shape)
```

```
[1 0 7 4 6 3 5 9 3 6]
(10,)
```

-Agglomerative clustering을 필두로 center 생성을 시도한 코드이다. 필자가 원하는 데이터 shape은 (10, 784)였지만 (10,)행태로 만들어졌다. 때문에 이대로 학습 시키면 다음과 같은 Error가 생성되었다.

```
ValueError: Expected 2D array, got 1D array instead:
array=[1 0 7 4 6 3 5 9 3 6].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape(1, -1) if it contains
a single sample.
```

-배열의 차원이 맞지 않아 생기는 Error였다.

```
print(agg)
```

```
[1 1 1 1 1 1 1 1 1 1 2 1 1 5 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2
 1 5 1 5 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 2 1 1 1 1 1 1
 1 1 1 1 1 7 1 1 1 1 1 1 1 3 5 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 7 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 3 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 2 7 2 7 7 7 7 7 7 7 7 8 7 7 4 7 7 7 7 7 7 7 0
 7 7 7 7 7 7 7 7 4 7 9 7 7 7 7 7 7 7 7 7 7 4 7 8 7 7 7 7 7 7 7 7 7 7 7 7 7 7
 7 4 7 7 7 7 7 7 7 4 2 7 4 7 7 7 7 7 7 7 7 7 7 4 7 7 7 7 7 7 7 7 7 7 7 4 7 7
 7 7 7 7 3 4 4 4 3 3 3 4 4 4 4 4 3 4 4 4 3 4 4 3 3 4 4 4 3 4 4 4 4 3 3 4 4
 3 4 4 4 4 8 3 8 4 4 4 3 4 3 4 3 7 3 4 3 4 4 4 3 4 4 4 3 7 4 3 5 3 3 3 4 4 4
 4 4 3 4 4 4 4 4 9 2 3 4 4 4 3 2 3 3 4 4 4 4 4 3 4 3 3 4 6 8 4 6 6 6 6 8 8 8
 6 9 6 8 6 8 6 8 8 0 8 6 8 6 6 6 6 6 6 8 6 8 6 8 6 8 6 9 6 6 8 6 8 6 8 6 6
 6 6 8 8 3 6 6 8 8 8 6 8 6 7 6 6 8 8 9 6 9 8 6 8 8 6 8 6 5 8 7 8 8 8 8 6 6
 6 6 6 8 8 8 9 6 8 6 8 6 8 9 6 8 8 6 6 3 3 2 4 3 2 4 4 3 2 2 2 2 4 3 4 9 3
 3 7 4 3 3 4 3 4 4 2 9 3 2 7 2 3 2 3 8 2 3 2 5 2 3 3 4 4 2 2 4 4 8 2 3 6 3
 3 2 3 2 2 3 2 3 3 3 3 7 4 3 4 4 2 3 2 4 2 3 2 2 3 2 2 3 2 3 3 2 3 8 3 4 2 3
 2 3 4 4 2 2 3 2 5 5 2 5 2 5 5 2 5 2 5 2 5 2 5 5 2 5 2 2 2 5 2 5 2 2 2 5 2 2 5 2 5
 5 5 5 5 5 5 2 5 5 5 5 2 5 5 2 5 5 2 2 2 5 5 5 5 5 2 5 2 5 2 5 2 2 2 2 5 5 5 5 2 2 5 2
 5 2 2 5 5 5 2 5 4 5 2 5 2 2 2 2 2 5 2 5 5 5 5 5 5 2 5 5 2 2 2 5 5 5 5 5 2 2 0 9 9
 9 9 6 9 9 9 9 9 9 9 9 9 9 9 8 9 9 8 9 9 9 9 9 9 9 9 9 9 8 9 9 9 9 9 9 9 9 9 9
 8 9 9 8 9 9 9 6 8 9 0 9 9 9 6 9 9 9 9 9 9 9 9 9 8 9 9 9 9 9 8 9 9 9 9 9 8 9 9 9 9 8 6
 9 9 8 9 0 9 8 9 9 9 9 9 9 9 9 6 8 9 9 9 9 8 9 8 0 8 8 3 3 4 3 8 8 8 3 3 4 2
 3 3 4 4 3 8 3 4 2 3 2 4 0 3 3 3 3 3 3 0 3 3 8 0 3 3 4 3 3 3 8 3 2 3 3 8
 0 6 3 4 3 8 2 3 3 2 8 2 3 4 3 3 4 2 2 8 3 4 4 8 3 3 3 3 3 3 0 3 3 2 4 3 3
 8 0 2 4 4 3 3 3 3 3 8 8 6 6 6 8 8 6 6 6 6 9 8 2 6 6 6 9 8 7 6 8 8 6 6 8 8 9
 9 6 6 6 6 9 8 8 6 9 9 6 6 6 8 6 8 8 6 8 6 6 6 8 6 8 9 8 6 8 8 9 8 8 6 6 6
 6 8 8 9 8 6 8 8 6 6 6 8 6 8 8 6 6 6 8 6 9 6 6 6 8 6 9 8 6 8 9 6 6 6 6 6 6
 8]
```

-1000개로 랜덤 추출한 X_train을 Agglomerative clustering에 fit_predict하고, print해본 결과 다음과 같이 군집화 된 predict결과 형태가 도출됐다.

	-결국 X_train과 군집화 된 predict결과가 mapping되는 index를 찾지 못했다. 그렇기에 label된 결과 값만 알뿐 결과 값을 만들어준 mapping되는 X_train 배열을 찾지 못한 것을 실패요인이라고 분석했다.