기계학습개론 과제3 리포트

202011047 김승태

1. Run the code below and arrange the results.

https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

코드를 위 주소에 있는 그대로 돌리려고 하였으나, 코드에 오류가 있음을 보아, 수정하고 시작하였다. 환경은 google colab에서 진행하였으며, 먼저 다음과 같이 install하였다.

```
!pip install hdbscan
!pip install -U scikit-learn
!pip install matplotlib
```

그 후, 다음과 같이 import하였는데, 기존 코드와 다른 점은, import hdbscan을 하였다는 점이다.

```
import time
import warnings
from itertools import cycle, islice

import matplotlib.pyplot as plt
import numpy as np

from sklearn import cluster, datasets, mixture
from sklearn.neighbors import kneighbors_graph
from sklearn.preprocessing import StandardScaler
import hdbscan
```

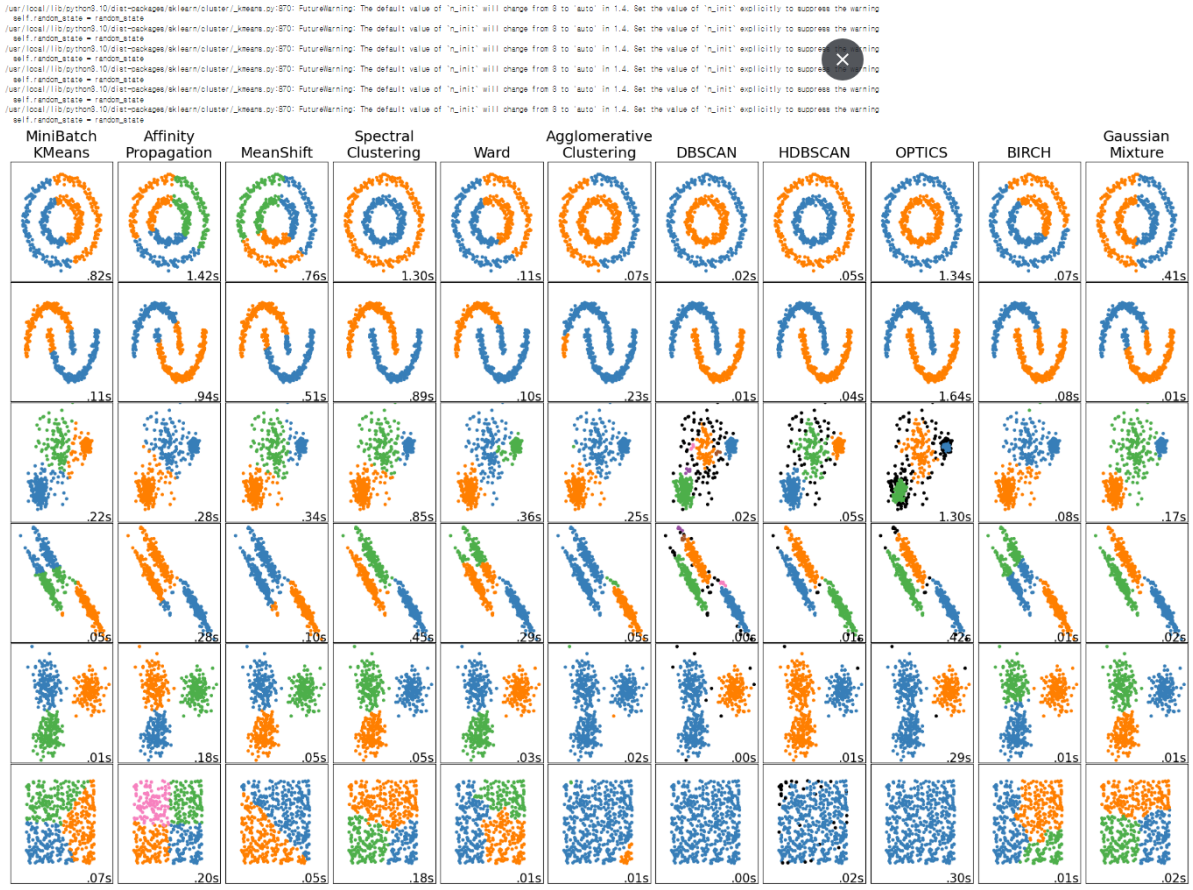
그 이유는, 다음과 같은 곳을 수정하였기 때문이다. 이전 코드로 하면, 에러가 날 수밖에 없다.

```
dbscan = cluster.DBSCAN(eps=params["eps"])
hdbscan_clust = hdbscan.HDBSCAN(
    min_samples=params["hdbscan_min_samples"],
    min_cluster_size=params["hdbscan_min_cluster_size"],
    allow_single_cluster=params["allow_single_cluster"],
)
```

위가 현재 코드이다. 아래는 이전 코드이다.

```
dbscan = cluster.DBSCAN(eps=params["eps"])
hdbscan = cluster.HDBSCAN(
    min_samples=params["hdbscan_min_samples"],
    min_cluster_size=params["hdbscan_min_cluster_size"],
    allow_single_cluster=params["allow_single_cluster"],
)
```

이전 코드로 하면, hdbscan = cluster.HDBSCAN에서 error가 나는데, 이는 cluster에서 HDBSCAN을 불러올 수 없기 때문이다. 그리하여, hdbscan을 직접 import하여 수행하였다. 결과는 다음과 같다. Futurewarning이 뜨긴 했지만, default value값만 변경되는 것이어서, 크게 문제는 없었다.



여기서 보게 되면, Spectral Clustering이 전반적으로 잘 된다는 것을 볼 수 있다.

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

아래는, 각 0~9클래스에서, 100개씩 랜덤으로 뽑아 오고, 이것을 3,4,5,6,7,8에서도 적용할 수 있도록 seed를 지정하였다. 코드는 다음과 같다.

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Set the number of samples per class
n_samples_per_class = 100
n_classes = 10
seed = 42  # Seed for reproducibility

# Initialize lists to hold sampled images and labels
sampled_images = []
sampled_labels = []

# Iterate over each class and sample images
for class_label in range(n_classes):
    # Get all images and labels for the current class
    class_images = x_train[y_train == class_label]
    class_labels = y_train[y_train == class_label]

    # Set the seed for numpy's random number generator before sampling
    np.random.seed(seed + class_label)  # Each class with a unique seed

    # Randomly sample n_samples_per_class images from the current class
    sampled_indices = np.random.choice(len(class_images), n_samples_per_class, replace=False)
    sampled_images.append(class_images[sampled_indices])
    sampled_labels.append(class_labels[sampled_indices])

# Convert the lists to numpy arrays
sampled_images = np.concatenate(sampled_images, axis=0)
sampled_labels = np.concatenate(sampled_labels, axis=0)

# Shuffle the sampled dataset
np.random.seed(seed)
shuffled_indices = np.random.permutation(len(sampled_labels))
sampled_images = sampled_images[shuffled_indices]
sampled_labels = sampled_labels[shuffled_indices]

# Flatten the images for clustering
n_samples, width, height = sampled_images.shape
flattened_images = sampled_images.reshape(n_samples, width * height)

# Standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
flattened_images_scaled = scaler.fit_transform(flattened_images)

# Verify the shape of the sampled data
print("Sampled images shape:", sampled_images.shape)  # Should be (1000, 28, 28)
print("Sampled labels shape:", sampled_labels.shape)  # Should be (1000,)
print("First 10 sampled labels:", sampled_labels[:10])
```

결과값은 다음과 같다.

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Sampled images shape: (1000, 28, 28)
Sampled labels shape: (1000,)
First 10 sampled labels: [5 7 7 6 4 6 6 5 8 1]
```

처음 10 샘플 데이터는 이와 같다.

3. For 1000 images, perform Agglomerative clustering, k-means clustering, Gaussian mixture model, Spectral clustering. (i.e., k=10)

```python
from sklearn.cluster import AgglomerativeClustering, KMeans, SpectralClustering
from sklearn.mixture import GaussianMixture

# Define clustering algorithms
clustering_algorithms = {
    'Agglomerative Clustering': AgglomerativeClustering(n_clusters=10),
    'KMeans': KMeans(n_clusters=10, random_state=seed),
    'Gaussian Mixture': GaussianMixture(n_components=10, random_state=seed),
    'Spectral Clustering': SpectralClustering(n_clusters=10, affinity='nearest_neighbors', random_state=seed)
}

# Dictionary to hold clustering results
clustering_results = {}

# Apply each clustering algorithm
for name, algorithm in clustering_algorithms.items():
    if name == 'Gaussian Mixture':
        clusters = algorithm.fit_predict(flattened_images_scaled)
    else:
        clusters = algorithm.fit_predict(flattened_images_scaled)
    clustering_results[name] = clusters
```

3번은 일단, 먼저 이렇게 k값은 10 으로 지정하여서 진행하였기 때문에 Aggolmerative Clustering: n_clusters=10, KMeans: n_clusters=10, Gaussian Mixture: n_components=10, Spectral Clustering: n_clusters=10 으로 코드를 제작하였다. 성공적으로 실행되었다. 결과값은 따로 나오지 않게 설계하였다. (perform만 하라고 하였으므로)

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

코드 이미지가 밀려, 아래에 제시하고, 먼저 결과를 설명하도록 하겠다. Rand index는 두 데이터 클러스터링 간의 유사성을 측정하며, 샘플 쌍을 고려해 예측 클러스터링과 실제 클러스터링에서 동일하게 또는 다르게 할당된 쌍을 계산한다. 이 코드의 결과에서는, Rand index : 0.2230 으로 클러스터링과 실제 레이블 간의 중간 수준의 일치를 보여주고, KMeans는 0.2615 으로 Agglomerative Clustering보다 약간 더 나은 성능을 보이고, Gaussian Mixture는 Kmeans와 동일한 성능을 보인다. 마지막으로 Spectral Clustering은 테스트한 방법 중 가장 높은 값은 0.2722값을 보여주어, Spectral Clustering이 가장 잘 된 것을 볼 수 있다. Mutual information 또한 Spectral Clustering이 가장 높은 값을 보여주며, 실제 레이블과의 상호 정보를 가장 잘 포착한다는 것을 알 수 있다. 이는 1번에서 보았던, Spectral Clustering이 여기에서도 잘 됨을 알 수 있다.

```python
from sklearn.metrics import adjusted_rand_score, adjusted_mutual_info_score

# Dictionary to hold evaluation results
evaluation_results = {}

# Compute evaluation metrics for each clustering result
for name, clusters in clustering_results.items():
    rand_index = adjusted_rand_score(sampled_labels, clusters)
    mutual_info = adjusted_mutual_info_score(sampled_labels, clusters)
    evaluation_results[name] = {
        'Rand Index': rand_index,
        'Mutual Information': mutual_info
    }

# Print evaluation results
for name, scores in evaluation_results.items():
    print(f"{name}:")
    print(f"  Rand Index: {scores['Rand Index']}")
    print(f"  Mutual Information: {scores['Mutual Information']}")

```

```
Agglomerative Clustering:
  Rand Index: 0.2230234494774759
  Mutual Information: 0.42992496438479966
KMeans:
  Rand Index: 0.2615445518615087
  Mutual Information: 0.396079408056459
Gaussian Mixture:
  Rand Index: 0.2615445518615087
  Mutual Information: 0.396079408056459
Spectral Clustering:
  Rand Index: 0.2721517473014819
  Mutual Information: 0.5207744591728529
```

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.

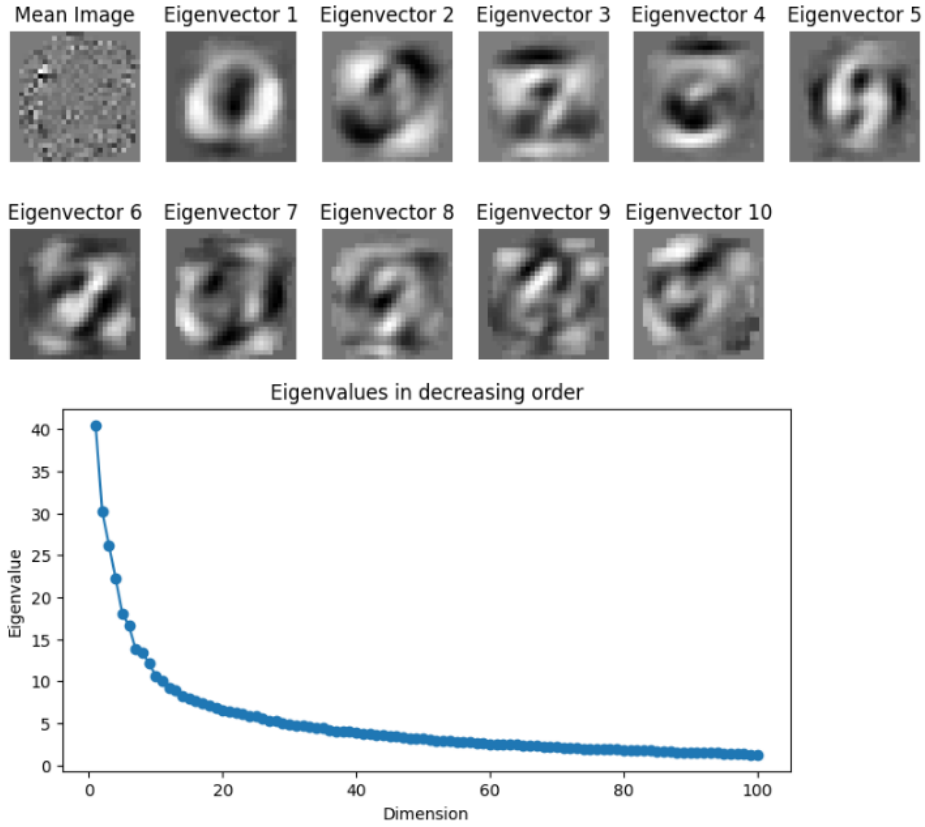
이 또한, 이미지가 페이지에 밀려, 결론을 먼저 설명하려고 한다. 각각의 결과는 이미지에 포함되어 있고, 그것들의 분석은 다음과 같다. Agglomerative Clustering은 데이터의 계층적 구조를 기반으로 클러스터를 형성하는데, MNIST 데이터의 복잡한 패턴을 제대로 포착하지 못해서 낮은 정확도를 보이고 있다. KMeans는 클러스터 중심을 반복적으로 조정해 데이터 포인트를 할당하는데, 그래서 MNIST 데이터의 고차원 특성을 반영해서 비교적으로 높은 정확도를 달성하였던 것으로 보인다. Gaussian Mixture도 데이터의 군집 특성을 잘 포착했다. Spectral Clustering은 다소 낮은 정확도를 보였으나, 이는 데이터의 국부적 구조를 잘 포착했지만, 전체적 분류 성능이 떨어진 것을 보여준다. 결과적으로, 여기에서의 Spectral Clustering은 데이터 지역적 특성을 잘 반영했지만, 전반적인 분류 성능은 떨어졌다는 것을 알 수 있다. 코드와 결과 이미지는 다음과 같다.

```python
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import accuracy_score

# Function to find cluster centers
def find_cluster_centers(clusters, data):
    centers = []
    for cluster_id in np.unique(clusters):
        cluster_points = data[clusters == cluster_id]
        center = cluster_points.mean(axis=0)
        centers.append(center)
    return np.array(centers)

# Perform 1-NN classification using cluster centers
for name, clusters in clustering_results.items():
    # Find cluster centers
    cluster_centers = find_cluster_centers(clusters, flattened_images_scaled)

    # Fit 1-NN classifier on cluster centers
    nn_classifier = NearestNeighbors(n_neighbors=1)
    nn_classifier.fit(cluster_centers)

    # Flatten and standardize the test images
    flattened_test_images = x_test.reshape(x_test.shape[0], -1)
    flattened_test_images_scaled = scaler.transform(flattened_test_images)

    # Find nearest cluster centers for each test image
    distances, indices = nn_classifier.kneighbors(flattened_test_images_scaled)

    # Predict labels based on nearest cluster centers
    predicted_labels = clusters[indices].flatten()

    # Compute accuracy
    accuracy = accuracy_score(y_test, predicted_labels)
    print(f"{name} 1-NN Classification Accuracy: {accuracy}")

```

```
Agglomerative Clustering 1-NN Classification Accuracy: 0.0744
KMeans 1-NN Classification Accuracy: 0.1896
Gaussian Mixture 1-NN Classification Accuracy: 0.1896
Spectral Clustering 1-NN Classification Accuracy: 0.1496
```

6. Run the PCA and Kernel PCA functions on the 1000 images used in (3). Plot the mean image and the first 10 eigenvectors (as images). Plot the eigenvalues (in decreasing order) as a function of dimension. Describe what you find in both plots. You can check the relevant codes for PCA and

Kernel PCA at:



첫 번째 플롯은 PCA를 수행한 결과 얻은 평균 이미지와 첫 10개의 고유 벡터를 보여준다. 평균 이미지는 모든 샘플 이미지의 평균을 나타내고, MINST 데이터 셋의 평균적인 숫자 이미지를 보여주며, 특정 숫자가 뚜렷하게 보이지 않고 흐릿하게 나타난다. 고유 벡터들은 첫 번째 고유 벡터가 데이터셋에서 가장 많은 변동성을 설명하며, 이후의 고유 벡터들은 점점 적은 변동성을 설명하게 된다. 그 결과, eigenvalue들은 첫 번째 주성분에서 시작해서 점점 작아지고, 몇몇 초기 주성분들이 데이터 대부분의 분산을 설명하기 때문에, 이후 주성분들의 기여가 급격히 감소하기 때문에 이런 그래프를 볼 수 있다.

7. Run K-means clustering on the reduced features using the PCA and kernel PCA, respectively. Compute “Rand index” and “mutual information based score” on the training data. Explain your findings.

결과는 다음과 같다.

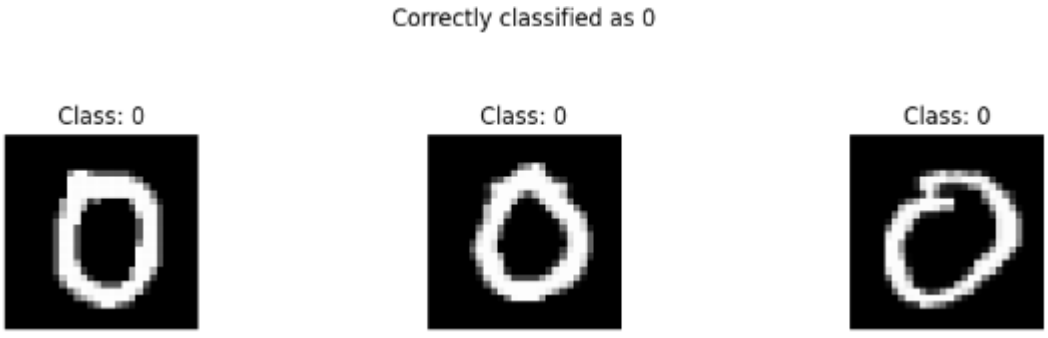
```
K-means Clustering on PCA-reduced features:
Adjusted Rand Index: 0.2462978846474394
Normalized Mutual Information Score: 0.3912855665220264

K-means Clustering on Kernel PCA-reduced features:
Adjusted Rand Index: 0.0013778227410522871
Normalized Mutual Information Score: 0.04908062982641528
```

PCA를 사용한 K-means 클러스터링에서 볼 수 있는 것은, PCA를 통해 차원을 축소한 후 K-means 클러스터링을 수행하게 되면, 데이터의 주요 변동성을 잘 반영하며 비교적 의미 있는 클러스터를 형성할 수 있다는 것이다. 그러나, Kernel PCA를 사용한 K-means 클러스터링의 경우, 매우 낮은 일치도를 보였는데, 이는 Kernel PCA가 데이터의 비선형 변환을 통해 차원을 축소하지만, K-means 클러스터링이 이러한 비선형 변환 후의 데이터 구조를 잘 포착하지 못하였음을 의미하는 것 같다.

7. Classify the MNIST test data set using 1-NN classifier and provide accuracy and visualize 3 correctly classified and 3 incorrectly classified images for each class. Explain your findings.

결과는 다음과 같다.

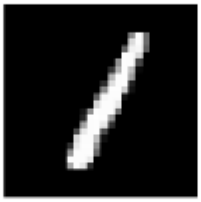
```
Accuracy: 0.8450
```



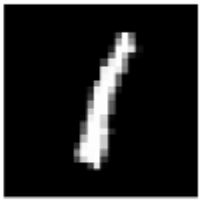
Correctly classified as 1

Class: 1

Class: 1



Class: 1



Correctly classified as 2

Class: 2

Class: 2

Class: 2

Incorrectly classified as 2

True: 2
Pred: 4

True: 2
Pred: 7

True: 2
Pred: 7

Class: 3

Class: 3

Class: 3

Incorrectly classified as 3

True: 3
Pred: 1

True: 3
Pred: 9

True: 3
Pred: 7

Class: 5

Class: 5

Class: 5

Incorrectly classified as 5

True: 5
Pred: 3

True: 5
Pred: 9

True: 5
Pred: 3

Correctly classified as 6

Class: 6

Class: 6

Class: 6

Incorrectly classified as 6

True: 6
Pred: 0

True: 6
Pred: 0

Correctly classified as 7

Class: 7

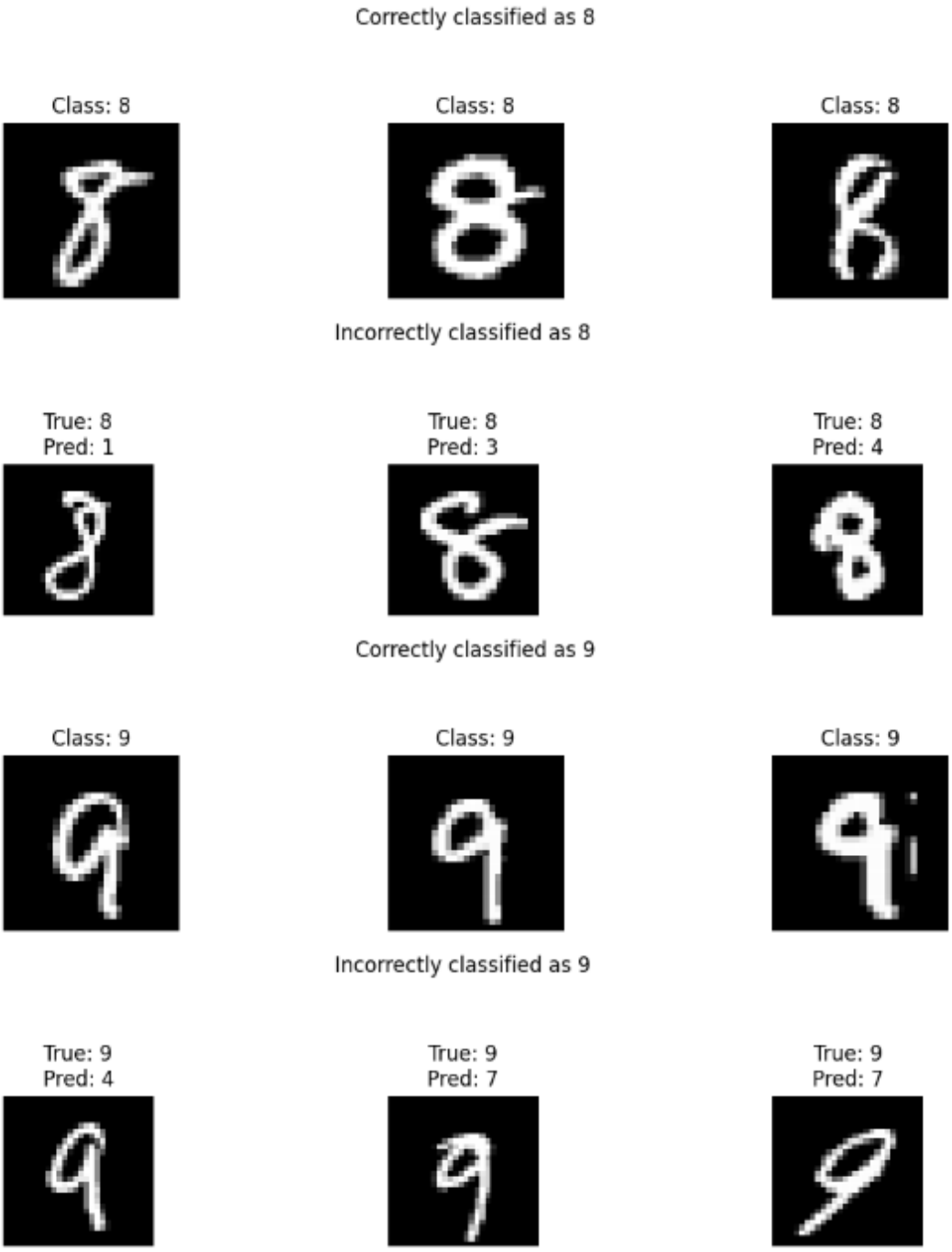
Class: 7

Class: 7

Incorrectly classified as 7

True: 7
Pred: 9

True: 7
Pred: 9



정확도는 84.5%로 굉장히 높았고, 이는 1-NN classifier가 MNIST 데이터 셋의 대부분의 숫자를 올바르게 분류했다는 것이다. 이미지들을 확인해 보면, 굉장히 흥미로운 경우들이 보인다. 0은 먼저 6과 헷갈려 하는 것 같았고, 이는, 실제로도 비슷한 숫자들이다. 1은 너무 명료하였던 탓인지, 전부 correctly하게 classified되었고, 2는 4, 7과 헷갈려하였으며, 3은 1,7,9와, 4는 7,9와, 5는 3,9와, 6은 0과, 7은 9와, 8은 1,3,4와, 9는 4,7과 주로 헷갈려 하는 것을 확인할 수 있었다. 높은 정확도라는 것이 먼저 놀랍고, 헷갈린 데이터들은 다 각기 헷갈려하는 숫자와의 공통 연관성을 눈으로 찾아볼 수 있는 것들이 많았다. 정확도가 높긴 하지만, 완전 만족스러울 정도는 아니므로, 추후에는 특징 추출 및 전처리나, 고급 분류 알고리즘을 사용하거나, 앙상블 방법을 이용하는 것이 좋겠다.