Machine Learning HW3

202211167 장지원

[Setting]

1. **Arrange the Result**

   해당 링크의 코드를 활용해 result를 준비해 보았다.

   Comparing different clustering algorithms on toy datasets — scikit-learn 1.4.2 documentation

2. **Preparing sample images**

   다음 코드를 이용해서 MNIST dataset에서 100의 sample 이미지를 뽑고, train, test set을 나누어 보았다.

```
for digit in range(10):
    indices = np.where(y == digit)[0]
    selected_indices.extend(np.random.choice(indices, num_samples_per_class,
replace=False))

selected_indices = np.array(selected_indices)  # Convert to numpy array
X_selected = X.iloc[selected_indices].values  # Ensure X_selected is a numpy array
y_selected = y.iloc[selected_indices].values  # Ensure y_selected is a numpy array

# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y_selected,
test_size=0.2, random_state=42)
```

[Clustering]

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

   밑과 같이 각 clustering model들을 정의해 보았다.

```
# Agglomerative Clustering
agglo = AgglomerativeClustering(n_clusters=10)
agglo_pred = agglo.fit_predict(X_test)

# K-Means Clustering
kmeans = KMeans(n_clusters=10, random_state=42)
kmeans_pred = kmeans.fit_predict(X_test)

# Gaussian Mixture Model
gmm = GaussianMixture(n_components=10, random_state=42)
gmm_pred = gmm.fit_predict(X_test)

# Spectral Clustering
spectral = SpectralClustering(n_clusters=10, random_state=42,
assign_labels='discretize')
spectral_pred = spectral.fit_predict(X_test)
```

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

   a. Rand index (adjust)

$$R = \frac{a+b}{\binom{2}{n}}$$

Eq. Rand index

Rand index는 가지고 있는 모든 '순서가 정해지지 않은' 데이터 쌍, $\binom{2}{n}$ 에 대하여 두 clustering(y_true, y_pred)의 결과에서 서로 동일한 cluster에 속하는 쌍의 개수와 동일하지 않은 cluster에 속하는 쌍의 개수의 확률을 구한 값이다. 이는 0과 1 사이의 값을 갖고, 1로 갈수록 정확한 분류를 했다는 것을 의미한다. 하지만 이 때 cluster 수가 많아지면 동일하지 않은 cluster에 속하는 쌍의 개수가 많아져서 clustering의 정확도와 상관없이 rand index가 커진다는 문제점이 있다. 이를 해결하기 위해 제안된 것이 adjust rand index이다.

$$R = \frac{\sum_{ij}\binom{n_{ij}}{2} - \sum_i\binom{a_i}{2}\sum_j\binom{b_j}{2}/\binom{2}{n}}{\frac{1}{2}\left[\sum_i\binom{a_i}{2} + \sum_j\binom{b_j}{2}\right] - \left[\sum_i\binom{a_i}{2}\sum_j\binom{b_j}{2}\right]/\binom{2}{n}}$$

Eq. Adjust rand index

Adjust rand index는 cluster의 개수까지 고려하기 때문에 신뢰도가 더 올라간다. 0.2를 기준으로 clustering이 잘 되었는지 평가한다.

b. Mutual info score (adjust)

$$MI = \sum_{i=1}^{r}\sum_{j=1}^{s} P(i,j)$$

Eq. Mutual info

Mutual info는 두 clustering 결과에서의 두 cluster 간 상호 의존성 값이다. 여기서도 마찬가지로 cluster 수가 많아지면 Mutual info가 커진다는 문제점이 있다. 이를 해결하기 위해서 Adjust Mutual info를 사용한다. 점수가 클수록 clustering이 잘 되었다고 할 수 있다.

결과는 밑과 같았다. Agglomerative Clustering, Kmeans Clustering, GMM Clustering에서는 괜찮은 결과를 얻을 수 있었지만, Spectral Clustering에서는 좋지 못한 결과를 얻었다. 우리는 이를 통해 Spectral Clustering은 MNIST dataset에 적합하지 않았음을 확인할 수 있다.

```
====  rand_score ====
Agglomerative Clustering Rand_score: 0.393
K-Means Clustering Rand_score: 0.262
Gaussian Mixture Model Rand_score: 0.262
Spectral Clustering Rand_score: 0.000

==== mutual_info_score ====
Agglomerative Clustering Mutual_info_score: 1.277
K-Means Clustering Mutual_info_score: 0.961
Gaussian Mixture Model Mutual_info_score: 0.961
Spectral Clustering Mutual_info_score: 0.035
```

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.

KNN과 Clustering은 '거리'를 기반으로 하여 데이터를 나타낸다는 점에서 유사하다. KNN은 K개의 이웃 데이터와 거리를 비교해서 classifying하고, Clustering은 center와의 거리 비교를 통해서 clustering한다. 따라서 동일한 데이터셋에서 1-NN classifier에서의 높은 정확도는 clustering에서 좋은 결과를 얻을 수 있다는 것을 의미한다.

결과는 밑과 같았다. **4.**에서 clustering 결과가 괜찮았듯이, 1-NN 정확도 역시 나쁘지 않은 것을 확인할 수 있다.

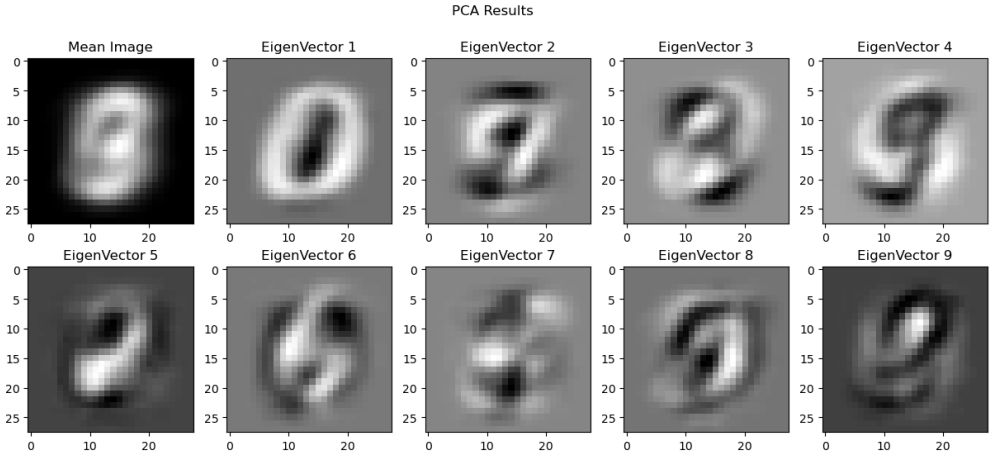
```
정확도: 84.00%
```

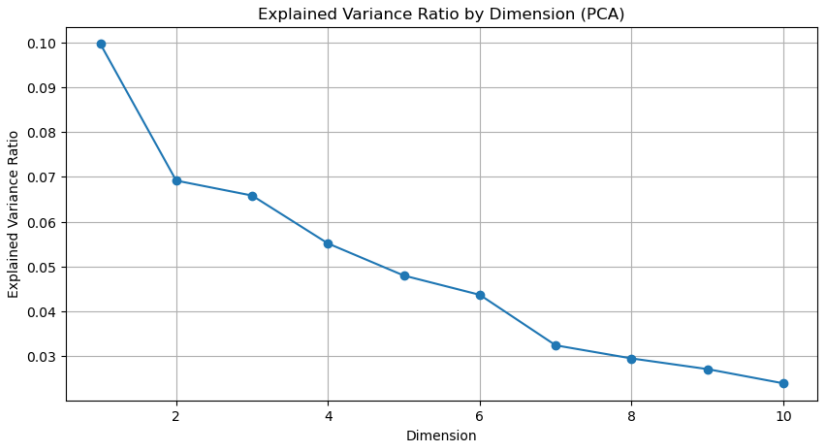
6. **Run the PCA and Kernel PCA functions on the 1000 images used in (3). Plot the mean image and the first 10 eigenvectors (as images). Plot the eigenvalues (in decreasing order) as a function of dimension.**

PCA는 차원 축소 방법이고, kernel PCA는 kernel 함수를 통해 고차원으로 mapping한 이후에 차원 축소를 진행하는 방법이다. 밑의 코드를 이용해서 PCA, 그리고 Kernel PCA 함수를 실행해 보았다.

```
# PCA 수행
pca = PCA(n_components=10)
X_pca = pca.fit_transform(X_selected)

# Kernel PCA 수행
kpca = KernelPCA(n_components=10, kernel='rbf')
X_kpca = kpca.fit_transform(X_selected)
```

다음 figure는 mean image를 plot한 결과와 eigenvalues(≈eigenvalues가 variance를 설명하는 정도)를 차원에 따른 함수로 나타낸 자료이다. 낮은 차원일수록 eigenvalue가 variance를 잘 설명하는 것을 확인할 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.**

PCA를 하면 기존 보다 정확도가 올라가고, kernel PCA를 사용하면 PCA를 사용했을 때 보다 더 높은 정확도를 얻을 수 있다.

```
PCA's score:
Rand score: 0.336
Mutual information score: 1.069

Kernel PCA's Score:
Rand score: 0.349
```

```
Mutual information score: 1.074
```

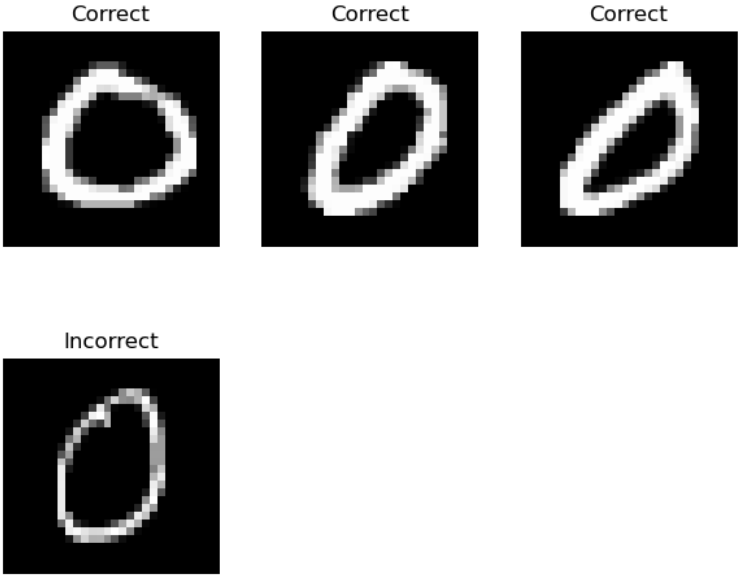
8. **Classify the MNIST test data set using 1-NN classifier and provide accuracy and visualize 3 correctly classified and 3 incorrectly classified images for each class. Explain your findings.**

1-NN을 실행해 보았고, 정답 index, 오답 index를 뽑아서 각각 세 개씩 plot하게 해보았다. 코드는 밑과 같이 작성하였다.

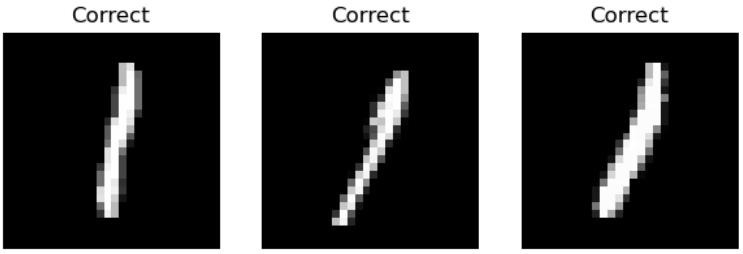
```
# 각 클래스별로 올바르게 분류된 이미지와 잘못 분류된 이미지
correct_indices = np.where(y_pred == y_test)[0]
incorrect_indices = np.where(y_pred != y_test)[0]
for digit in range(10):
    correct_digit_indices = correct_indices[np.where(y_test[correct_indices] ==
digit)]
    incorrect_digit_indices = incorrect_indices[np.where(y_test[incorrect_indices]
== digit)]

    plt.figure(figsize=(15, 6))
    plt.suptitle(f"Digit {digit}")

    # 올바르게 분류된 이미지 플롯
    for i, idx in enumerate(correct_digit_indices[:3]):
        plt.subplot(2, 6, i + 1)
        plt.imshow(X_test[idx].reshape(28, 28), cmap='gray')
        plt.title("Correct")
        plt.axis('off')

    # 잘못 분류된 이미지 플롯
    for i, idx in enumerate(incorrect_digit_indices[:3]):
        plt.subplot(2, 6, 6 + i + 1)
        plt.imshow(X_test[idx].reshape(28, 28), cmap='gray')
        plt.title("Incorrect")
        plt.axis('off')

    plt.show()
```

결과는 밑 그림과 같다. 0, 1, 7과 같이 단순한 형태의 숫자는 거의 다 맞추는 것을 확인할 수 있었고(세 숫자에 대해서는 incorrect가 3개 이하였다), 3, 4, 5등의 복잡한 숫자에 대해서는 정답을 맞추지 못하는 경우가 있었음을 확인할 수 있었다. 또한 얇은 글씨체들을 잘 인식하지 못하는 것을 확인할 수 있었다. 결과를 확인해 보면, correct data는 대부분 두꺼운 글씨체를 가지고 있는 걸 알 수 있다.

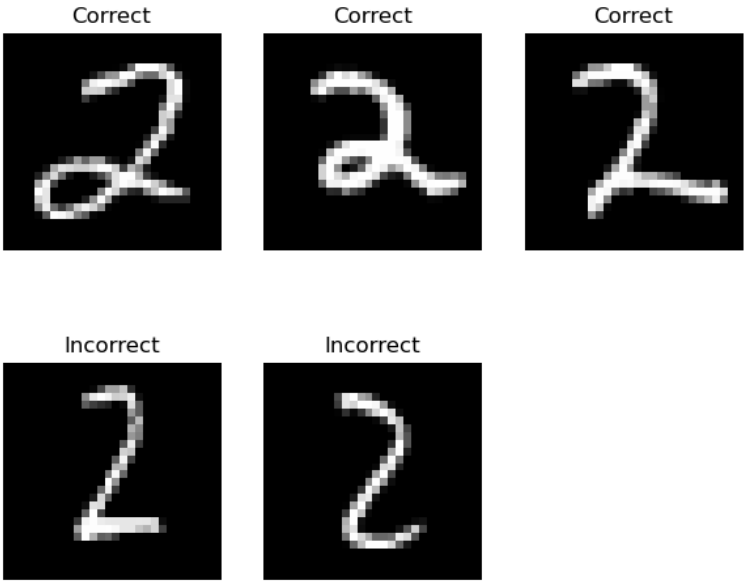
Digit 0



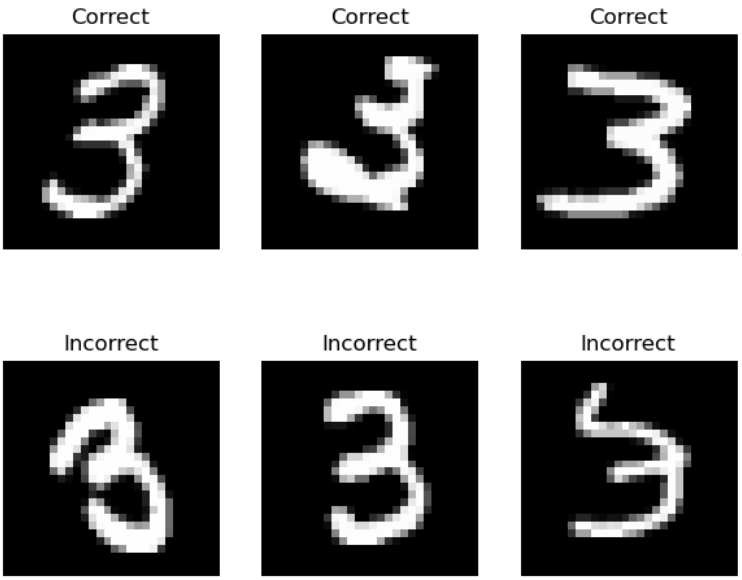
Digit 1



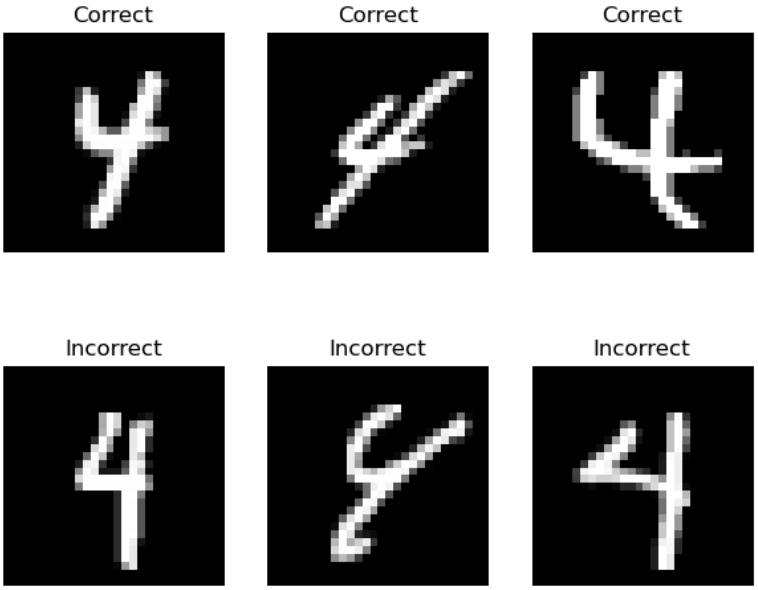
Digit 2



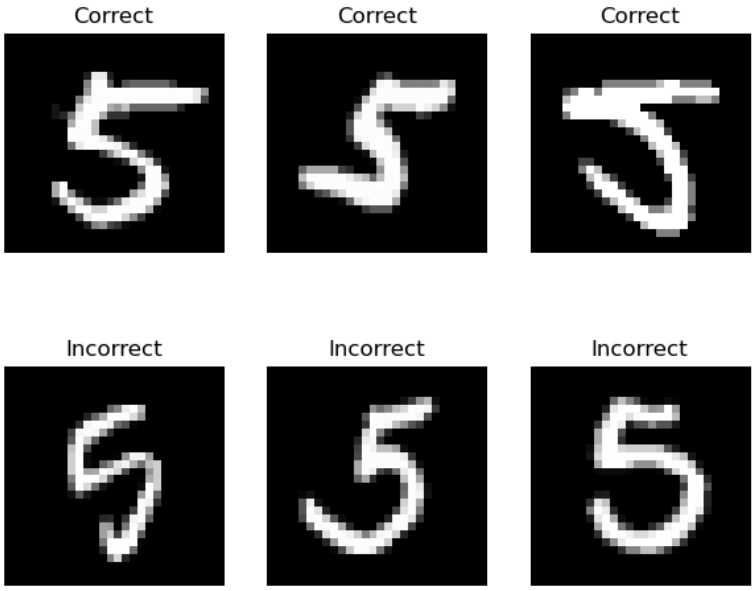
Digit 3



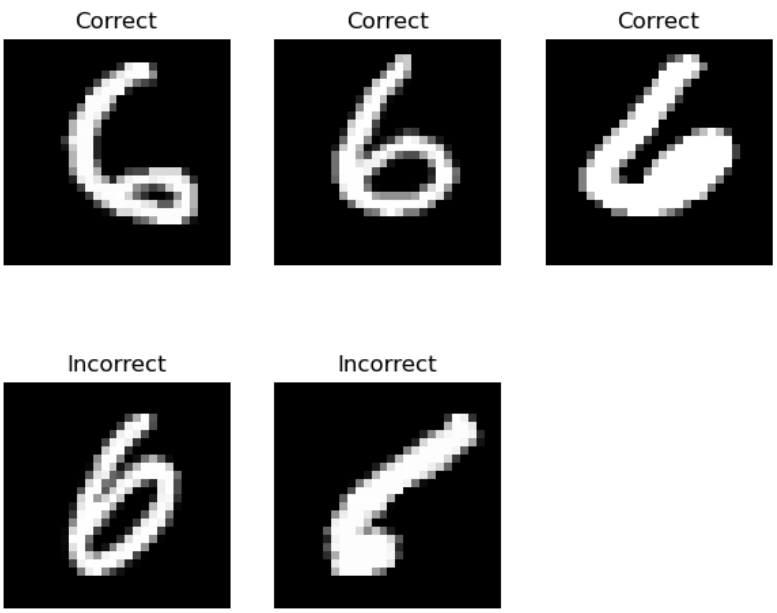
Digit 4



Digit 5



Digit 6

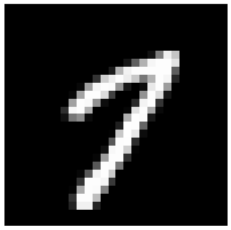


Digit 7

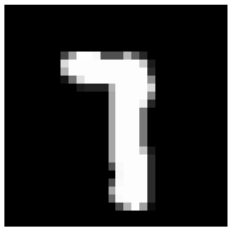
Correct

Correct

Correct

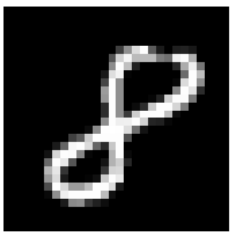


Incorrect



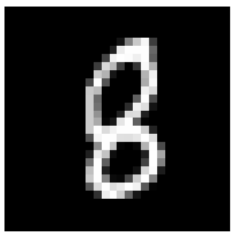
Digit 8

Correct



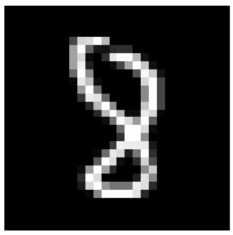
Correct

Correct

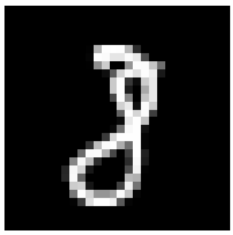


Incorrect

Incorrect



Incorrect



Digit 9