[CSE302] Introduction to Machine Learning

< Assignment 4 >

(Deadline: 2022-06-07)

202123008 Jinmin Kim (김진민)

Phone: 010-6266-6099

Mail: rlawlsals@dgist.ac.kr

(0) 데이터 불러오기 및 전처리

** Code

```python
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.datasets import fetch_openml
from sklearn.neighbors import KNeighborsClassifier
from sklearn import cluster
from sklearn.metrics import adjusted_rand_score
from sklearn.metrics import adjusted_mutual_info_score
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA

```

: 파이썬 모듈 임포트

```python
# ###################################### Data Importing ###################################### #
print('** Data importing')
print('')
# MNIST 데이터 불러오기
mnist = fetch_openml('mnist_784', version=1)
print(mnist.keys())
X, y = mnist['data'], mnist['target']
X = np.asarray(X)
y = np.asarray(y).astype(int)

print('MNIST data shape: ', X.shape, 'MNIST label shape: ', y.shape)

```

: MNIST 데이터셋을 불러옴

```python
# 각 숫자에 해당하는 레이블의 인덱스를 분류
for k in range(10):
    globals()['label{}'.format(k)] = []

for i in range(len(y)):
    for k in range(10):
        if y[i] == k: globals()['label{}'.format(k)].append(i)

for k in range(10):
    print('label {} length : '.format(k), len(globals()['label{}'.format(k)]))

# 각 레이블의 인덱스를 랜덤하게 100개씩 추출
for k in range(10):
    globals()['label{}_100_idx'.format(k)] = random.sample(globals()['label{}'.format(k)], 100)
    print('sampled label {} length : '.format(k), len(globals()['label{}_100_idx'.format(k)]))

# 100개씩 추출된 각 레이블 인덱스를 저장
for k in range(10):
    globals()['data{}_100'.format(k)] = np.zeros(shape=(100, 784))
    for s in range(100):
        globals()['data{}_100'.format(k)][s] = X[globals()['label{}_100_idx'.format(k)][s]]
    print('sampled data {} shape: '.format(k), globals()['data{}_100'.format(k)].shape)

```

: 각 숫자에 해당하는 레이블의 인덱스를 분류

: 분류된 인덱스를 각 숫자마다 랜덤하게 100개씩 추출

: 각 숫자의 인덱스를 저장해놓고, 해당 인덱스에 대한 데이터를 불러올 예정.

```python
# 숫자 2에 대한 100개의 데이터 저장
X_train_2 = globals()['data{}_100'.format(2)]
y_train_idx_2 = globals()['label{}_100_idx'.format(2)]
# print('Digit-2 training data shape :', X_train_2.shape)
# print('Ground truth label : \n', y[y_train_idx_2])

# 숫자 0~9에 대한 1000개의 데이터 저장
X_train_09 = 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_09 = 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)]])

```

: 숫자 2에 대한 데이터와 숫자 0~9에 대한 데이터를 저장.

(1) Run the PCA and kernel PCA functions on the Digit-2-Space (100 training images used in Assignment 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.

** Code

```
# ######################################## Problem 1 ######################################## #
print('** Problem 1 : Run the PCA and kernel PCA functions on the Digit-2-Space')
print('')

# Mean image 추출
X_train_mean_2 = np.mean(X_train_2, axis=0)
print(X_train_mean_2.shape)
rec_X_train_mean_2 = X_train_mean_2.reshape(28, 28)

# PCA
pca_2 = PCA(n_components=10)
pca_2.fit(X_train_2)
eigen_vec_0 = pca_2.components_
print('(PCA) eigenvector shape: ', eigen_vec_0.shape)

# Kernel PCA (kernel= 'linear')
kpca_1 = KernelPCA(n_components=10, kernel='linear')
kpca_1.fit(X_train_2.T)
eigen_vec_1 = kpca_1.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_1.shape)

# Kernel PCA (kernel= 'poly')
kpca_2 = KernelPCA(n_components=10, kernel='poly')
kpca_2.fit(X_train_2.T)
eigen_vec_2 = kpca_2.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_2.shape)

# Kernel PCA (kernel= 'rbf')
kpca_3 = KernelPCA(n_components=10, kernel='rbf')
kpca_3.fit(X_train_2.T)
eigen_vec_3 = kpca_3.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_3.shape)

# Kernel PCA (kernel= 'sigmoid')
kpca_4 = KernelPCA(n_components=10, kernel='sigmoid')
kpca_4.fit(X_train_2.T)
eigen_vec_4 = kpca_4.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_4.shape)

# Kernel PCA (kernel= 'cosine')
kpca_5 = KernelPCA(n_components=10, kernel='cosine')
kpca_5.fit(X_train_2.T)
eigen_vec_5 = kpca_5.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_5.shape)
```

: 1. Mean image : 숫자 2에 대한 이미지 100개의 평균값을 구해 생성함.
: 2. PCA : scikit-learn의 PCA 모듈을 통해 생성함.
: 3~7. Kernel PCA : scikit-learn의 KernelPCA 모듈을 통해 생성함.
Kernel은 scikit-learn에서 제공하는 'linear', 'poly', 'rbf', 'sigmoid', 'cosine'을 사용하였음.

```
** Problem 1 : Run the PCA and kernel PCA functions on the Digit-2-Space

(784,)
(PCA) eigenvector shape:  (10, 784)
(Kernel PCA) eigenvector shape:  (10, 784)
(Kernel PCA) eigenvector shape:  (10, 784)
(Kernel PCA) eigenvector shape:  (10, 784)
(Kernel PCA) eigenvector shape:  (10, 784)
```

: PCA를 사용하여 (100, 784)의 크기를 가지는 데이터로부터 (10, 784)의 크기를 가지는 Eigenvector 10개를 추출하였음.
: 상기 Eigenvector에 기존 데이터를 프로젝션한다면 (100, 10)의 크기를 가지도록 차원을 축소할 수 있음. (784개 -> 10개)

** Results

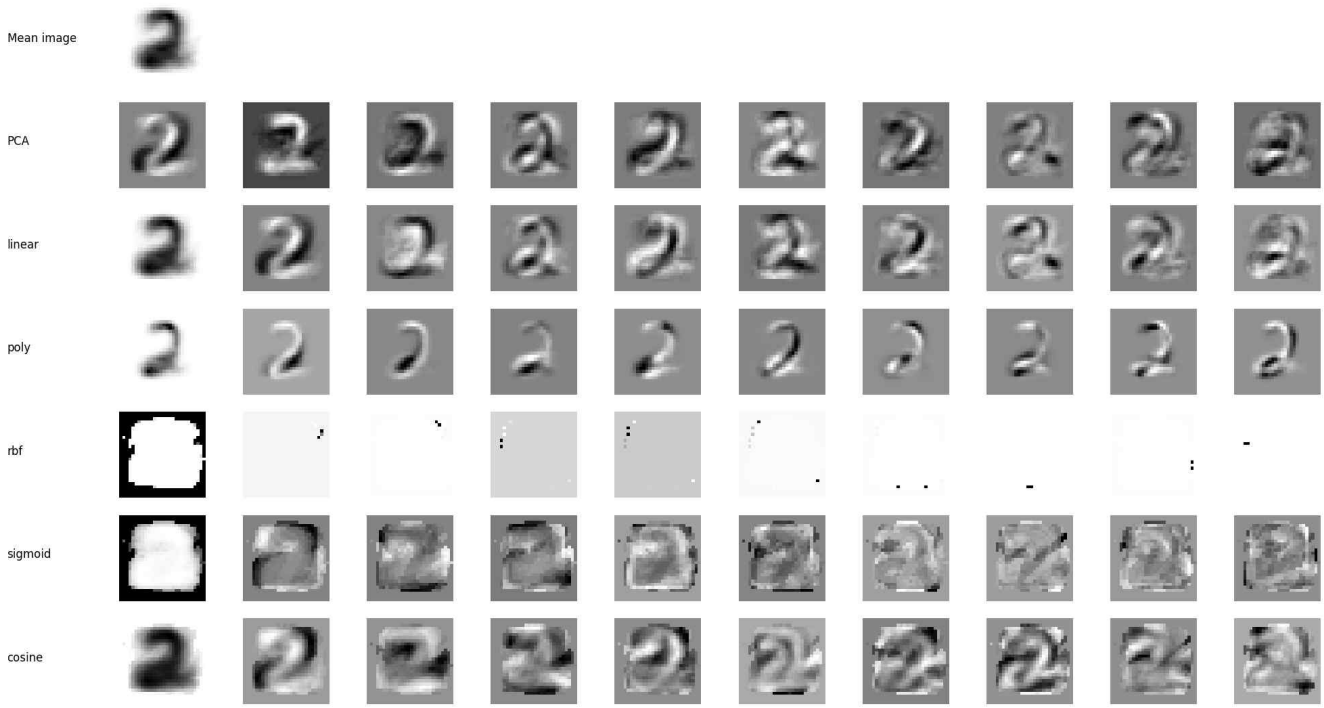


그림 1. 숫자 2의 Eigenvector를 시각화한 결과

: (왼쪽) 1번째 eigenvector ~ (오른쪽) 10번째 eigenvector를 뜻함.

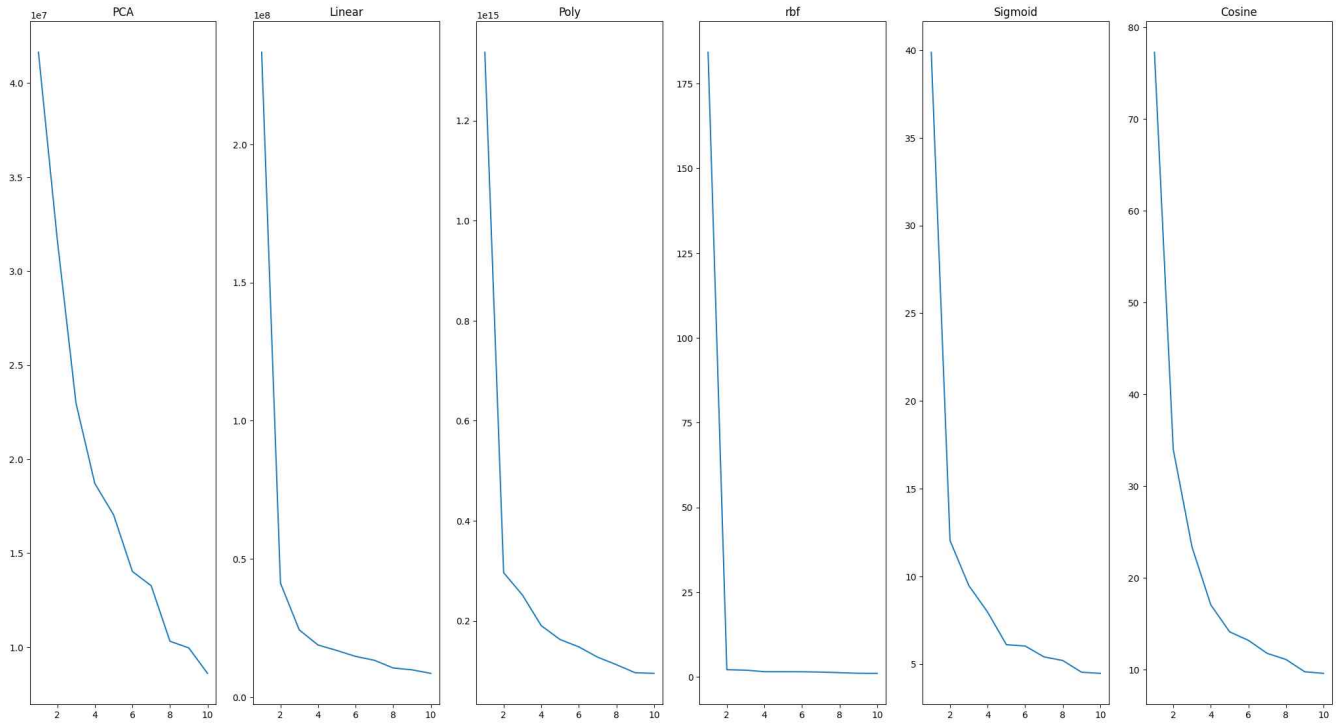


그림 2. 숫자 2의 Eigenvalue를 플롯한 결과

** Discussion

- 그림 1을 통해 알 수 있는 점은, Kernel을 사용하지 않았을 때보다 Kernel을 사용하였을 때 더 특징을 선명하게 잡아낼 수 있는 것으로 보인다. 하지만 어떤 Kernel을 사용하는지에 따라 효과가 매우 달랐다. 'linear', 'poly' 커널은 커널을 사용하지 않은 PCA보다 더 숫자 2에 가까운 Eigenvector를 얻을 수 있었다. 그런데 'rbf', 'sigmoid', 'cosine' 커널을 사용하였을 때는 오히려 더 인식하기 어려운 결과가 나왔다.

- 그림 2를 통해 이 결과를 조금 더 정량적으로 확인할 수 있는데, 모든 결과에서 첫 번째 eigenvalue가 가장 높게 나왔지만 그 스케일이 다르다. 커널을 사용하지 않은 PCA와 'linear', 'poly' 커널을 사용한 PCA에서는 각각 10^7, 10^8, 10^{15} 로 매우 높은 값이 추출되었다. 육안으로 보더라도 'poly' 커널을 사용한 PCA에서 숫자가 가장 잘 구별되는 것처럼 보인다.

(2) Run the PCA and kernel PCA functions on all 1000 training images used in Assignment 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. Compare these plots to the ones you created in (1).

** Code

```
# ######################################## Problem 2 ######################################## #
print('** Problem 2 : Run the PCA and kernel PCA functions on all 1000 training images')
print('')

# Mean image 추출
X_train_mean_09 = np.mean(X_train_09, axis=0)
print(X_train_mean_09.shape)
rec_X_train_mean_09 = X_train_mean_09.reshape(28, 28)

# PCA
pca_09 = PCA(n_components=10)
pca_09.fit(X_train_09)
eigen_vec_09_0 = pca_09.components_
print('(PCA) eigenvector shape: ', eigen_vec_09_0.shape)

# Kernel PCA (kernel= 'linear')
kpca_09_1 = KernelPCA(n_components=10, kernel='linear')
kpca_09_1.fit(X_train_09.T)
eigen_vec_09_1 = kpca_09_1.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_09_1.shape)

# Kernel PCA (kernel= 'poly')
kpca_09_2 = KernelPCA(n_components=10, kernel='poly')
kpca_09_2.fit(X_train_09.T)
eigen_vec_09_2 = kpca_09_2.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_09_2.shape)

# Kernel PCA (kernel= 'rbf')
kpca_09_3 = KernelPCA(n_components=10, kernel='rbf')
kpca_09_3.fit(X_train_09.T)
eigen_vec_09_3 = kpca_09_3.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_09_3.shape)

# Kernel PCA (kernel= 'sigmoid')
kpca_09_4 = KernelPCA(n_components=10, kernel='sigmoid')
kpca_09_4.fit(X_train_09.T)
eigen_vec_09_4 = kpca_09_4.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_09_4.shape)

# Kernel PCA (kernel= 'cosine')
kpca_09_5 = KernelPCA(n_components=10, kernel='cosine')
kpca_09_5.fit(X_train_09.T)
eigen_vec_09_5 = kpca_09_5.eigenvectors_.T
print('(Kernel PCA) eigenvector shape: ', eigen_vec_09_5.shape)

```

: 문제 1과 같은 방식으로 숫자 0~9의 데이터를 이용하여 PCA를 진행함.

** Results

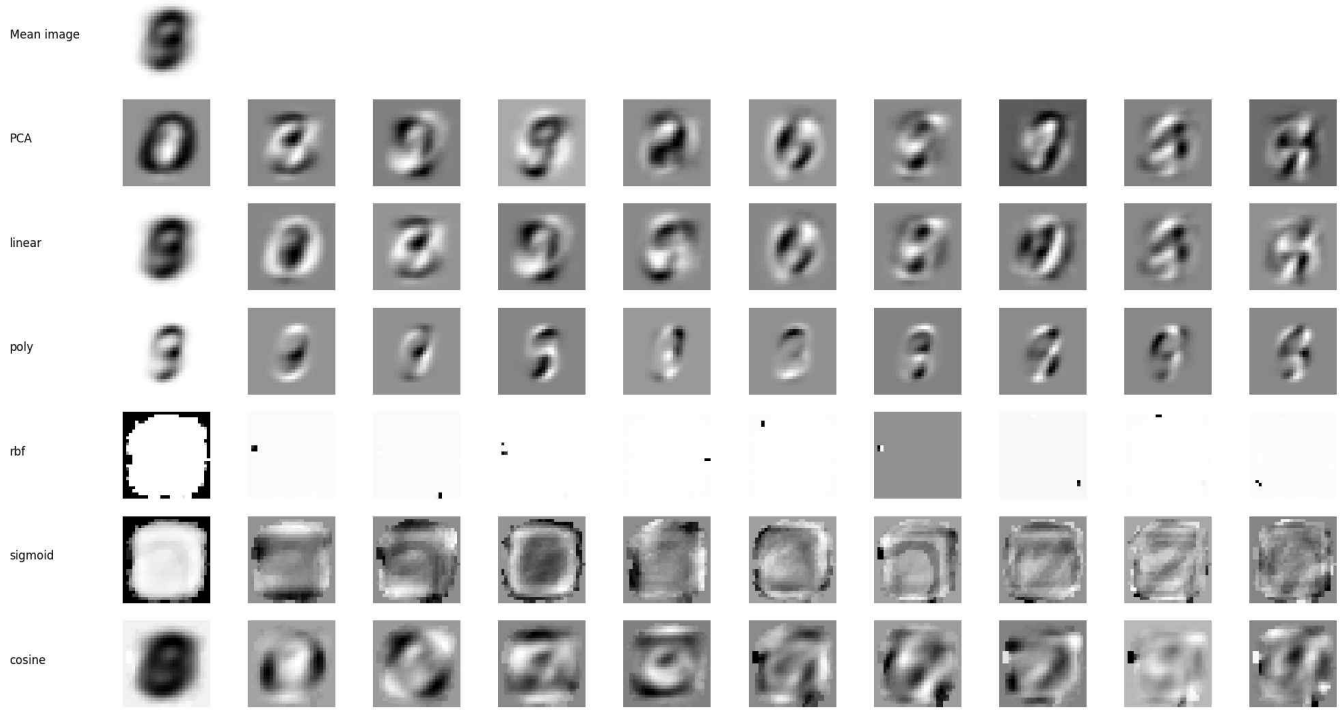


그림 3. 숫자 0~9의 Eigenvector를 시각화한 결과

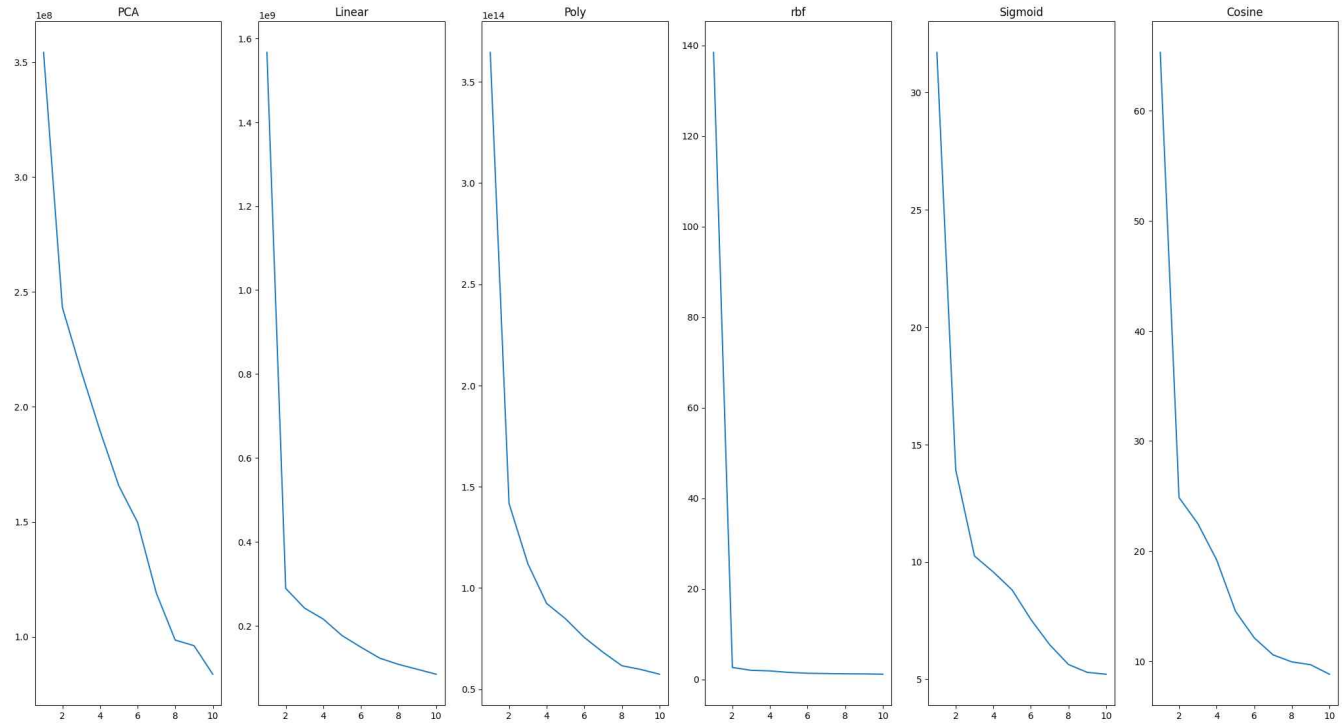


그림 4. 숫자 0~9의 Eigenvalue를 플롯한 결과

** Discussion

- 문제 1에서는 숫자 2에 대해서만 PCA를 진행하였었고, 문제 2에서는 숫자 0~9의 데이터에 대한 PCA를 진행하였다. ‘rbf’, ‘sigmoid’, ‘cosine’ 커널을 사용하였을 때 문제 1에서보다 더 숫자의 특징을 인식하기가 어려워진 것으로 보인다.
- 0~9의 모든 숫자가 다 겹쳐 있어서 무슨 숫자인지 구별하기는 어려운 것 같다.

(3) 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.

** Code

```
# PCA를 통해 추출한 eigenvector에 원본 트레이닝 데이터를 프로젝션
pca_proj_09 = projection(eigen_vec_09_0, X_train_09)
kpca_1_proj_09 = projection(eigen_vec_09_1, X_train_09)
kpca_2_proj_09 = projection(eigen_vec_09_2, X_train_09)
kpca_3_proj_09 = projection(eigen_vec_09_3, X_train_09)
kpca_4_proj_09 = projection(eigen_vec_09_4, X_train_09)
kpca_5_proj_09 = projection(eigen_vec_09_5, X_train_09)
print('Projection with 10 eigenvectors of pca (shape): ', pca_proj_09.shape)
print('Projection with 10 eigenvectors of linear kernel pca (shape): ', kpca_1_proj_09.shape)
print('Projection with 10 eigenvectors of poly kernel pca (shape): ', kpca_2_proj_09.shape)
print('Projection with 10 eigenvectors of rbf kernel pca (shape): ', kpca_3_proj_09.shape)
print('Projection with 10 eigenvectors of sigmoid kernel pca (shape): ', kpca_4_proj_09.shape)
print('Projection with 10 eigenvectors of cosine kernel pca (shape): ', kpca_5_proj_09.shape)

```

: 차원축소된 데이터를 얻기 위해, 문제 2에서 추출한 eigenvector에 원본 데이터를 프로젝션함.

```
Projection with 10 eigenvectors of pca (shape):  (1000, 10)
Projection with 10 eigenvectors of linear kernel pca (shape):  (1000, 10)
Projection with 10 eigenvectors of poly kernel pca (shape):  (1000, 10)
Projection with 10 eigenvectors of rbf kernel pca (shape):  (1000, 10)
Projection with 10 eigenvectors of sigmoid kernel pca (shape):  (1000, 10)
Projection with 10 eigenvectors of cosine kernel pca (shape):  (1000, 10)
```

: (1000, 784) -> (1000, 10)으로 차원축소된 모습을 볼 수 있음.

```
# 프로젝션된 데이터를 kmeans로 분류
kmeans_pca = cluster.KMeans(n_clusters=10)
kmeans_kpca_1 = cluster.KMeans(n_clusters=10)
kmeans_kpca_2 = cluster.KMeans(n_clusters=10)
kmeans_kpca_3 = cluster.KMeans(n_clusters=10)
kmeans_kpca_4 = cluster.KMeans(n_clusters=10)
kmeans_kpca_5 = cluster.KMeans(n_clusters=10)
kmeans_labels_pca = kmeans_pca.fit_predict(pca_proj_09)
kmeans_labels_kpca_1 = kmeans_kpca_1.fit_predict(kpca_1_proj_09)
kmeans_labels_kpca_2 = kmeans_kpca_2.fit_predict(kpca_2_proj_09)
kmeans_labels_kpca_3 = kmeans_kpca_3.fit_predict(kpca_3_proj_09)
kmeans_labels_kpca_4 = kmeans_kpca_4.fit_predict(kpca_4_proj_09)
kmeans_labels_kpca_5 = kmeans_kpca_5.fit_predict(kpca_5_proj_09)

```

: 차원축소된 데이터를 kmeans clustering 모델에 피팅시킨다.

```
401     # Kmeans의 label과 True label을 매칭시키기
402     ground_truth = y[y_train_idx_09]
403     kmeans_labels_pca = matching_label(ground_truth, kmeans_labels_pca)
404     kmeans_labels_kpca_1 = matching_label(ground_truth, kmeans_labels_kpca_1)
405     kmeans_labels_kpca_2 = matching_label(ground_truth, kmeans_labels_kpca_2)
406     kmeans_labels_kpca_3 = matching_label(ground_truth, kmeans_labels_kpca_3)
407     kmeans_labels_kpca_4 = matching_label(ground_truth, kmeans_labels_kpca_4)
408     kmeans_labels_kpca_5 = matching_label(ground_truth, kmeans_labels_kpca_5)
409
```

: kmeans clustering을 통해 클러스터링된 레이블은 실제 레이블과 일치하지 않는다. 따라서 가장 빈번하게 발생한 숫자를 해당 레이블로 매치시켜준다.

```
410     # Rand index, Mutual information score 계산
411     print('')
412     print('* 1. Rand index')
413     print('')
414     rand_score_pca = adjusted_rand_score(ground_truth, kmeans_labels_pca)
415     rand_score_kpca_1 = adjusted_rand_score(ground_truth, kmeans_labels_kpca_1)
416     rand_score_kpca_2 = adjusted_rand_score(ground_truth, kmeans_labels_kpca_2)
417     rand_score_kpca_3 = adjusted_rand_score(ground_truth, kmeans_labels_kpca_3)
418     rand_score_kpca_4 = adjusted_rand_score(ground_truth, kmeans_labels_kpca_4)
419     rand_score_kpca_5 = adjusted_rand_score(ground_truth, kmeans_labels_kpca_5)
420     print('rand_score_pca :', rand_score_pca)
421     print('rand_score_kpca_1 :', rand_score_kpca_1)
422     print('rand_score_kpca_2 :', rand_score_kpca_2)
423     print('rand_score_kpca_3 :', rand_score_kpca_3)
424     print('rand_score_kpca_4 :', rand_score_kpca_4)
425     print('rand_score_kpca_5 :', rand_score_kpca_5)
426
427     print('')
428     print('* 2. Mutual information based score')
429     print('')
430     mutual_score_pca = adjusted_mutual_info_score(ground_truth, kmeans_labels_pca)
431     mutual_score_kpca_1 = adjusted_mutual_info_score(ground_truth, kmeans_labels_kpca_1)
432     mutual_score_kpca_2 = adjusted_mutual_info_score(ground_truth, kmeans_labels_kpca_2)
433     mutual_score_kpca_3 = adjusted_mutual_info_score(ground_truth, kmeans_labels_kpca_3)
434     mutual_score_kpca_4 = adjusted_mutual_info_score(ground_truth, kmeans_labels_kpca_4)
435     mutual_score_kpca_5 = adjusted_mutual_info_score(ground_truth, kmeans_labels_kpca_5)
436     print('mutual_score_pca :', mutual_score_pca)
437     print('mutual_score_kpca_1 :', mutual_score_kpca_1)
438     print('mutual_score_kpca_2 :', mutual_score_kpca_2)
439     print('mutual_score_kpca_3 :', mutual_score_kpca_3)
440     print('mutual_score_kpca_4 :', mutual_score_kpca_4)
441     print('mutual_score_kpca_5 :', mutual_score_kpca_5)
442
```

: Assignment 3에서 사용한 코드를 이용하여, Rand index와 Mutual information based score를 계산한다.

** Results

```
* 1. Rand index

rand_score_pca : 0.3487170839158473
rand_score_kpca_1 : 0.3689328193095136
rand_score_kpca_2 : 0.3242365972801024
rand_score_kpca_3 : 0.07431477521684994
rand_score_kpca_4 : 0.295573593574649
rand_score_kpca_5 : 0.31110523741037965

* 2. Mutual information based score

mutual_score_pca : 0.4741899049643283
mutual_score_kpca_1 : 0.4943537314518939
mutual_score_kpca_2 : 0.4440686783114654
mutual_score_kpca_3 : 0.10389165418398082
mutual_score_kpca_4 : 0.4139722135761006
mutual_score_kpca_5 : 0.4120971745239197
```

: 계산된 Rand index 및 Mutual information besed score

** Discussion

- 두 지표 모두 그렇게 좋은 성능을 나타내지는 못하였다.
- 해당 데이터는 0~9 숫자 모두에 대한 데이터 1000개를 통합하여 진행한 것인데, 해당 숫자 레이블과 PCA를 통해 생성되는 Eigenvector의 특징이 일치하지 않은 것으로 볼 수 있다.

(4) Classify the MNIST test data set using 1-NN classifier like Assignment 3. Compute accuracy scores and compare them with the accuracy in Assignment 3.

** Code

```
# ######################################## Problem 4 ######################################## #
print('** Problem 4 : Classifying the MNIST test data set using the center of each cluster')
print('')
# 테스트 데이터셋 생성 (앞에서부터 50000개의 데이터를 추출)
X_test = np.zeros(shape=(50000, 784))
y_test_idx = np.arange(50000)
for s in range(50000): X_test[s] = X[s]
y_test = y[y_test_idx]

```

: MNIST 데이터로부터 50,000개의 테스트 데이터셋을 추출한다.

```
# 테스트 데이터셋 차원 축소 ( 784 -> 10 )
X_test_proj_pca = projection(eigen_vec_09_0, X_test)
X_test_proj_kpca_1 = projection(eigen_vec_09_1, X_test)
X_test_proj_kpca_2 = projection(eigen_vec_09_2, X_test)
X_test_proj_kpca_3 = projection(eigen_vec_09_3, X_test)
X_test_proj_kpca_4 = projection(eigen_vec_09_4, X_test)
X_test_proj_kpca_5 = projection(eigen_vec_09_5, X_test)

```

: 문제 2에서 추출한 Eivenvector 10개에 테스트 데이터셋을 프로젝션하여 차원축소한다.

```
# Kmeans의 각 클러스터 센터를 추출
kmeans_center_pca = kmeans_pca.cluster_centers_
kmeans_center_kpca_1 = kmeans_kpca_1.cluster_centers_
kmeans_center_kpca_2 = kmeans_kpca_2.cluster_centers_
kmeans_center_kpca_3 = kmeans_kpca_3.cluster_centers_
kmeans_center_kpca_4 = kmeans_kpca_4.cluster_centers_
kmeans_center_kpca_5 = kmeans_kpca_5.cluster_centers_

```

: 문제 3에서 진행한 kmeans clustering의 클러스터 중심을 추출한다.

```
# 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)
kNN5 = KNeighborsClassifier(n_neighbors=1)
kNN6 = KNeighborsClassifier(n_neighbors=1)
kNN1.fit(kmeans_center_pca, np.arange(10))
kNN2.fit(kmeans_center_kpca_1, np.arange(10))
kNN3.fit(kmeans_center_kpca_2, np.arange(10))
kNN4.fit(kmeans_center_kpca_3, np.arange(10))
kNN5.fit(kmeans_center_kpca_4, np.arange(10))
kNN6.fit(kmeans_center_kpca_5, np.arange(10))

```

: Assignment 3에서 사용한 코드를 이용하여, 클러스터링 센터에 대한 Dicision boundary를 생성한다.

```
# 학습된 1-NN 모델을 통해 테스트 데이터셋을 예측
predict1 = kNN1.predict(X_test_proj_pca)
predict2 = kNN2.predict(X_test_proj_kpca_1)
predict3 = kNN3.predict(X_test_proj_kpca_2)
predict4 = kNN4.predict(X_test_proj_kpca_3)
predict5 = kNN5.predict(X_test_proj_kpca_4)
predict6 = kNN6.predict(X_test_proj_kpca_5)

# 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)
accuracy5 = accuracy_score(y_test, predict5)
accuracy6 = accuracy_score(y_test, predict6)
print('Accuracy of PCA using 1-NN : ', accuracy1)
print('Accuracy of Linear kernel PCA using 1-NN : ', accuracy2)
print('Accuracy of Poly kernel PCA center using 1-NN : ', accuracy3)
print('Accuracy of rbf kernel PCA using 1-NN : ', accuracy4)
print('Accuracy of Sigmoid kernel PCA using 1-NN : ', accuracy5)
print('Accuracy of Cosine kernel PCA using 1-NN : ', accuracy6)

```

: 생성된 1-NN 모델을 통해 차원축소된 테스트 데이터셋을 예측한다.
: 마지막으로 우리가 아는 레이블과 예측된 값 사이의 정확도를 출력한다.

** Results

```
Accuracy of PCA using 1-NN :  0.07434
Accuracy of Linear kernel PCA using 1-NN :  0.06934
Accuracy of Poly kernel PCA center using 1-NN :  0.16012
Accuracy of rbf kernel PCA using 1-NN :  0.07222
Accuracy of Sigmoid kernel PCA using 1-NN :  0.0212
Accuracy of Cosine kernel PCA using 1-NN :  0.06244
```

** Discussion

- 위 결과는 문제 3에서 생긴 오차가 누적된 것으로 보인다. 정확도를 평가하기에 무리가 있는 이유는 분류하려는 레이블과 eigenvector가 일치하지 않았기 때문으로 추정된다.

(5) Visualize 3 correctly classified and 3 incorrectly classified images for each class. Explain your findings.

** Code

```
# ######################################## Problem 5 ######################################## #
print('** Problem 5 : Visualize 3 correctly classified and 3 incorrectly classified images for each class')
print('')

fig3, axs3 = plt.subplots(8, 11)

# Extracting correct & incorrect images for each class
for i in range(10):
    globals()['cor_{}'.format(i)] = []
    globals()['incor_{}'.format(i)] = []

for i in range(50000):
    for j in range(10):
        if y_test[i] == j:
            if predict1[i] == j: globals()['cor_{}'.format(j)].append(i)
            else: globals()['incor_{}'.format(j)].append(i)

```

: 테스트 데이터에서 잘 구분된 이미지와 구분되지 않은 데이터를 분류하여 저장한다.

```
# Visualization of images
for i in range(10):
    if len(globals()['cor_{}'.format(i)]) >= 3:
        globals()['num{}_cor_{}'.format(i, 0)] = X_test[globals()['cor_{}'.format(i)][0]]
        globals()['num{}_cor_{}'.format(i, 1)] = X_test[globals()['cor_{}'.format(i)][1]]
        globals()['num{}_cor_{}'.format(i, 2)] = X_test[globals()['cor_{}'.format(i)][2]]
        test_digit_1 = globals()['num{}_cor_{}'.format(i, 0)]
        test_digit_2 = globals()['num{}_cor_{}'.format(i, 1)]
        test_digit_3 = globals()['num{}_cor_{}'.format(i, 2)]
        test_digit_image_1 = test_digit_1.reshape(28, 28)
        test_digit_image_2 = test_digit_2.reshape(28, 28)
        test_digit_image_3 = test_digit_3.reshape(28, 28)
        axs3[1, i + 1].imshow(test_digit_image_1, cmap="binary")
        axs3[2, i + 1].imshow(test_digit_image_2, cmap="binary")
        axs3[3, i + 1].imshow(test_digit_image_3, cmap="binary")

    if len(globals()['cor_{}'.format(i)]) == 2:
        globals()['num{}_cor_{}'.format(i, 0)] = X_test[globals()['cor_{}'.format(i)][0]]
        globals()['num{}_cor_{}'.format(i, 1)] = X_test[globals()['cor_{}'.format(i)][1]]
        test_digit_1 = globals()['num{}_cor_{}'.format(i, 0)]
        test_digit_2 = globals()['num{}_cor_{}'.format(i, 1)]
        test_digit_image_1 = test_digit_1.reshape(28, 28)
        test_digit_image_2 = test_digit_2.reshape(28, 28)
        axs3[1, i + 1].imshow(test_digit_image_1, cmap="binary")
        axs3[2, i + 1].imshow(test_digit_image_2, cmap="binary")

    if len(globals()['cor_{}'.format(i)]) == 1:
        globals()['num{}_cor_{}'.format(i, 0)] = X_test[globals()['cor_{}'.format(i)][0]]
        test_digit_1 = globals()['num{}_cor_{}'.format(i, 0)]
        test_digit_image_1 = test_digit_1.reshape(28, 28)
        axs3[1, i + 1].imshow(test_digit_image_1, cmap="binary")

    globals()['num{}_incor_{}'.format(i, 0)] = X_test[globals()['incor_{}'.format(i)][0]]
    globals()['num{}_incor_{}'.format(i, 1)] = X_test[globals()['incor_{}'.format(i)][1]]
    globals()['num{}_incor_{}'.format(i, 2)] = X_test[globals()['incor_{}'.format(i)][2]]
    test_digit_1 = globals()['num{}_incor_{}'.format(i, 0)]
    test_digit_2 = globals()['num{}_incor_{}'.format(i, 1)]
    test_digit_3 = globals()['num{}_incor_{}'.format(i, 2)]
    test_digit_image_1 = test_digit_1.reshape(28, 28)
    test_digit_image_2 = test_digit_2.reshape(28, 28)
    test_digit_image_3 = test_digit_3.reshape(28, 28)
    axs3[5, i + 1].imshow(test_digit_image_1, cmap="binary")
    axs3[6, i + 1].imshow(test_digit_image_2, cmap="binary")
    axs3[7, i + 1].imshow(test_digit_image_3, cmap="binary")

```

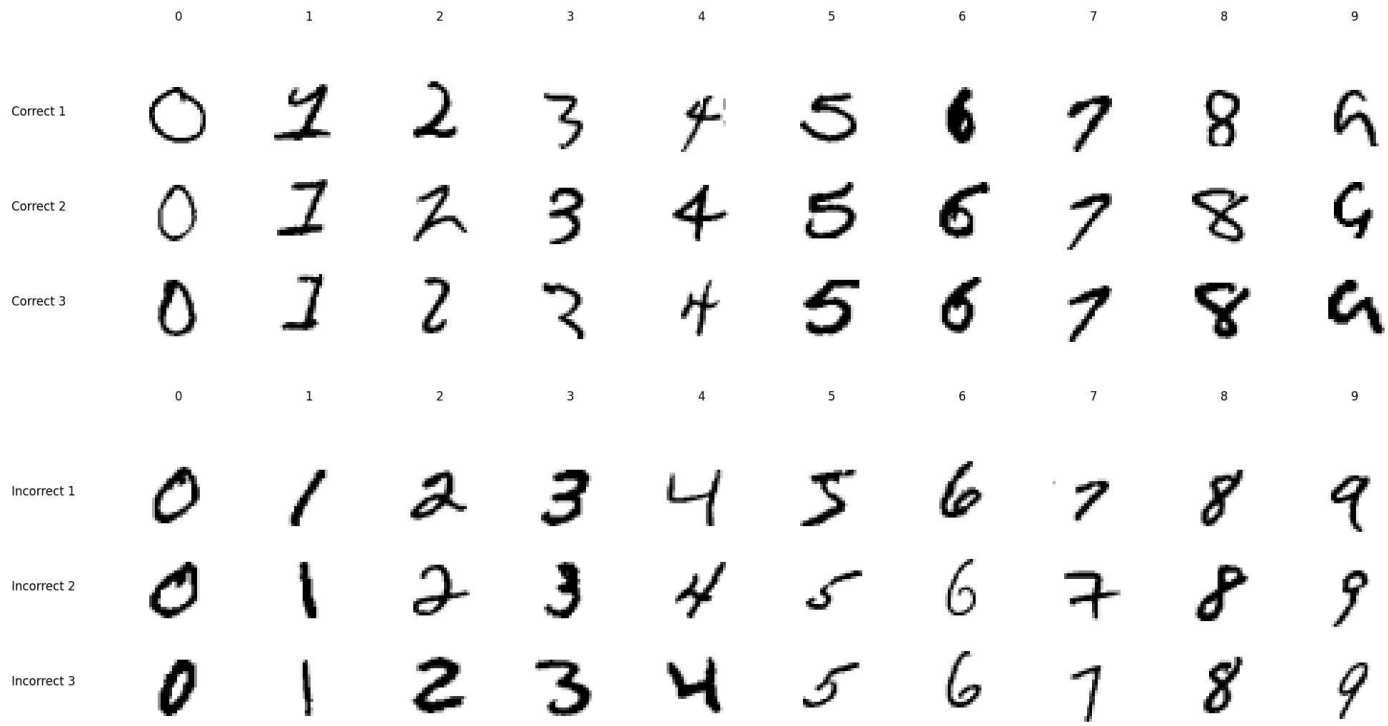
: 이미지를 시각화한다.

```
# Plot settings
axs3[1, 0].text(0.2, 0.5, 'Correct 1', fontsize='large')
axs3[2, 0].text(0.2, 0.5, 'Correct 2', fontsize='large')
axs3[3, 0].text(0.2, 0.5, 'Correct 3', fontsize='large')
axs3[5, 0].text(0.2, 0.5, 'Incorrect 1', fontsize='large')
axs3[6, 0].text(0.2, 0.5, 'Incorrect 2', fontsize='large')
axs3[7, 0].text(0.2, 0.5, 'Incorrect 3', fontsize='large')
for i in range(10):
    axs3[0, i+1].text(0.5, 0.5, '{}'.format(i), fontsize='large')
    axs3[4, i+1].text(0.5, 0.5, '{}'.format(i), fontsize='large')
for i in range(8):
    for j in range(11):
        axs3[i, j].axis("off")

plt.show()
```

: 이미지 세부 세팅 및 열기

** Results



: 각 숫자마다 잘 구분된 이미지 3개, 잘 구분되지 않은 이미지 3개를 시각화한 결과

** Discussion

- 잘 구분된 이미지와 잘 구분되지 않은 이미지들 육안으로의 차이가 있었다.
- 어떤 한 특징이 명확하고 비슷한 각도와 크기를 가지고 있는 데이터들은 잘 구분되었고, 해당 특징을 가지지 않은 데이터는 잘 구분되지 않은 것으로 보인다.

** 머신러닝은 전혀 다뤄보지 않았었지만 수업을 통해 scikit-learn이 익숙해진 것 같습니다. 결과를 깔끔하게 뽑아내지는 못했지만 앞으로 연구하면서 성공시켜보도록 하겠습니다. 한 학기 동안 많이 배워갑니다. 감사합니다.