CS302 Introduction to machine learning

School of Undergraduate Studies, DGIST

201611161 정원균

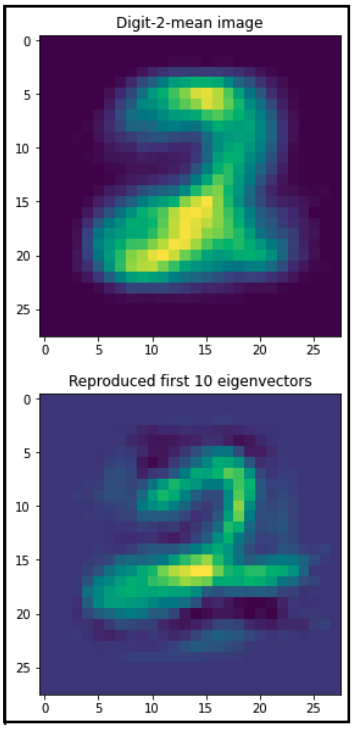
Homework report 4.

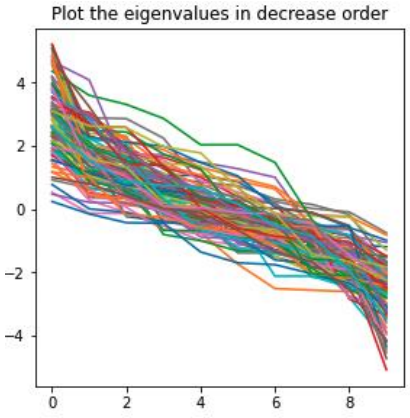
2022. 06. 07

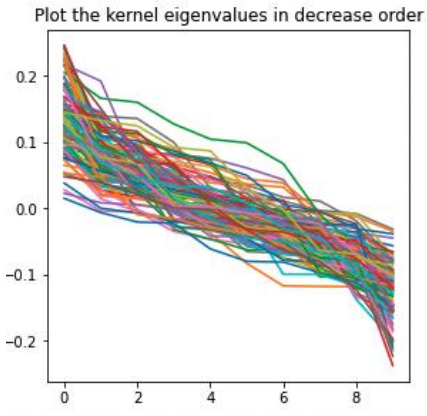
Q1. Run the PCA and kernel PCA functions on the Digit-2-Space (100 training images used in Assignment 3). Plot the mean image and the first 10 eigenvectors (as images). Plot the eigenvalues (in decreasing order) as a function of dimension. Describe what you find in both plots.

```python
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
X_train = X_train.reshape(60000,28,28,1)
X_test  = X_test.reshape(10000,28,28,1)

new_X_Train = np.reshape(X_train,(-1,28*28))
new_X_Test = np.reshape(X_test,(-1,28*28))

class_list  = [[] for i in range(10)]

for i in range(len(X_train)):
    class_list[y_train[i]].append(X_train[i])


for i in range(10):
    random.shuffle(class_list[i])
    sample_X = class_list[i][:100]
    class_list[i] = sample_X

class_list = np.array(class_list)
```

```python
pca = PCA(10)
kpca = KernelPCA(kernel = 'rbf',fit_inverse_transform=True, n_components= 10)

pca_model = pca.fit_transform(class_list[2].reshape(-1,784))
kpca_model = kpca.fit_transform(class_list[2].reshape(-1,784))


plt.figure(figsize = (15,15))
plt.subplot(2,2,1)
plt.imshow(np.array(class_list[2]).mean(axis = 0).reshape(28,28))
plt.title('Digit-2-mean image')
plt.show()

appx_pca = pca.inverse_transform(pca_model)
appx_kpca = kpca.inverse_transform(kpca_model)

plt.figure(figsize = (15,15))
plt.subplot(2,2,2)
plt.imshow(appx_pca[0].reshape(28,28))
plt.title("Reproduced first 10 eigenvectors")
plt.show()


sorted_pca_model = np.sort(pca_model,axis = 1)[:,::-1]
sorted_kpca_model = np.sort(kpca_model,axis = 1)[:,::-1]

plt.figure(figsize = (15,15))
plt.subplot(2,2,3)
for i in range(len(sorted_pca_model)):
  plt.plot(sorted_pca_model[i])
plt.title("Plot the eigenvalues in decrease order")
plt.show()

plt.figure(figsize = (15,15))
plt.subplot(2,2,4)
for i in range(len(sorted_kpca_model)):
  plt.plot(sorted_kpca_model[i])
plt.title("Plot the kernel eigenvalues in decrease order")
plt.show()
```



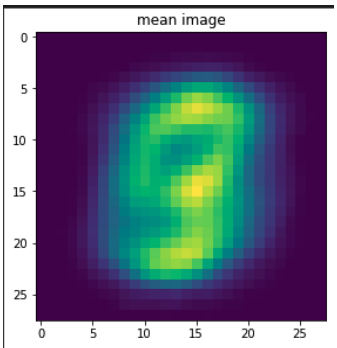


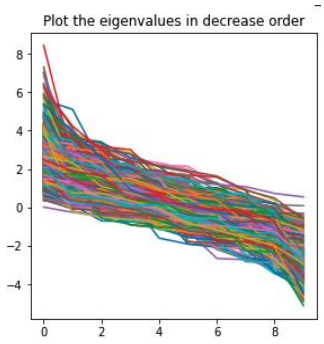


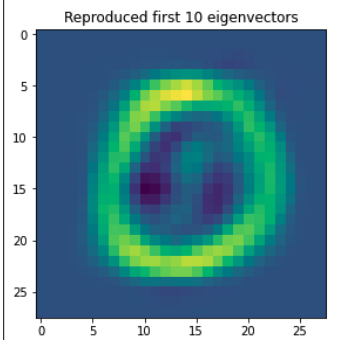
과제 3 번에서 사용하였던 Mnist 손글씨 데이터에서 각 클래스 별로 100 개씩 random sampling 한 총 1000 개의 데이터를 학습에 사용하였다. 1000 개의 학습데이터를 10 개의 component 로 PCA 와 'rbf' Kernel PCA 을 하였다. 숫자 2 에 해당하는 Mean image 와 10 개의 eigen vector 로 Dimension Reduction 을 한 다음에 복원한 이미지를 보여준다. Mean image 는 2-digit space 에 있는 2 에 해당하는 모든 image 들이 다 합쳐진 값의 평균이미지를 보여주기 때문에 두껍고 퍼진 2 를 나타낸다. 또 10 개의 eigen vector 로 차원을 줄이고 복원한 이미지는 10 개의 feature 들만 가지고 표현해야하기 때문에 조금 불분명한 이미지가 나오지만 충분히 2 를 표현함을 알 수 있다. PCA 와 Kernel PCA 을 통해서 얻은 Eigen value 들의 plot 을 그려보면 대체적으로 유사한 형태를 보인다는 것을 알 수 있다.

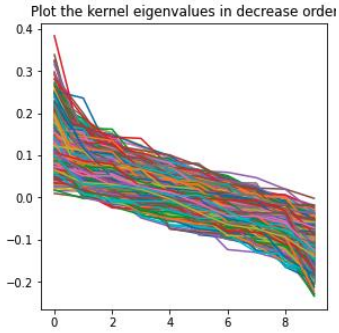
Q2. Run the PCA and kernel PCA functions on all 1000 training images used in Assignment 3. Plot the mean image and the first 10 eigenvectors (as images). Plot the eigenvalues (in decreasing order) as a function of dimension. Describe what you find in both plots. Compare these plots to the ones you created in (1).

```
pca = PCA(10)
kpca = KernelPCA(kernel = 'rbf',fit_inverse_transform=True, n_components= 10)

pca_model_2 = pca.fit_transform(np.array(class_list).reshape(-1,784))
kpca_model_2 = kpca.fit_transform(np.array(class_list).reshape(-1,784))

plt.figure(figsize = (10,10))

plt.subplot(2,2,1)
#plt.imshow(np.array(class_list).reshape(-1,784).mean(axis = 0).reshape(28,28))
plt.imshow(np.array(class_list).reshape(1000,-1).mean(axis = 0).reshape(28,28))
plt.title('mean image')
plt.show()


appx_pca_2 = pca.inverse_transform(pca_model_2)
appx_kpca_2 = kpca.inverse_transform(kpca_model_2)

plt.figure(figsize = (10,10))
plt.subplot(2,2,2)
plt.imshow(appx_pca_2[40].reshape(28,28))
plt.title("Reproduced first 10 eigenvectors")
plt.show()


sorted_pca_model_2 = np.sort(pca_model_2,axis = 1)[:,::-1]
sorted_kpca_model_2 = np.sort(kpca_model_2,axis = 1)[:,::-1]

plt.figure(figsize = (10,10))
plt.subplot(2,2,3)
for i in range(len(sorted_pca_model_2)):
  plt.plot(sorted_pca_model_2[i])
plt.title("Plot the eigenvalues in decrease order")
plt.show()

plt.figure(figsize = (10,10))
plt.subplot(2,2,4)
for i in range(len(sorted_kpca_model_2)):
  plt.plot(sorted_kpca_model_2[i])
plt.title("Plot the kernel eigenvalues in decrease order")
plt.show()
```









역시 1 번에서 Sampling 한 100 개씩 10 개 class 에 대해서 PCA 와 kernel PCA 을 적용하였다. Mean image 같은 경우에 0 부터 9 까지 모든 이미지들의 평균값을 나타내기 위해서 0,3,9 등등 다양한 숫자들이 섞여서 보임을 알 수 있다. 하지만 이를 10 개 eigen vector 로 Reduction 을 한 다음 복원한 결과 0 과 비슷한 이미지를 보인다. 이는 여러 이미지가 섞인 이미지에서 가장 weight 가 큰 주 성분만을 뽑아서 차원 축소를 한 뒤 다시 복원하기 때문에 mean image 에서 보이는 숫자의 태두리 모양이 적용이 크게 되어서 이 부분이 복원 시에 뚜렷하게 나타났다. 따라서 Q1 의 경우엔 실제 2 데이터를 복원한 이미지라고 할 수 있지만 Q2 에서는 복원한 이미지가 0 과 비슷하지만 실제로 0 이미지를 차원 축소 한 후 복원했다고 단정지을 수는 없다. 역시 PCA 와 Kernel PCA 를 통해서 구해진 eigen vector 을 plot 으로 나타내면 다음과 같다. 또한 PCA 를 적용시킨 data 수가 더 많기 때문에 Q1. 에서 보이는 plot 보다 더 빽빽한 plot 이 나온다는 것을 알 수 있고 PCA 와 kernel PCA 가 비슷한 형태를 보인다는 것 역시 알 수 있다.

Q3. 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.

```
from sklearn.metrics.cluster import rand_score
from sklearn.metrics import mutual_info_score
from sklearn.metrics import normalized_mutual_info_score


pca_model_R = pca.fit_transform(np.array(class_list).reshape(-1,784))
kpca_model_R = kpca.fit_transform(np.array(class_list).reshape(-1,784))

appx_pca_R = pca.inverse_transform(pca_model_R).reshape(-1,28*28)
appx_kpca_R = kpca.inverse_transform(kpca_model_R).reshape(-1,28*28)


kmeans = cluster.KMeans(n_clusters = 10).fit(appx_pca_R.reshape(-1,28*28))


temp_k = []
label_k = kmeans.predict(appx_pca_R.reshape(-1,28*28))
#print(label_k.reshape(10,-1))

for i in range(10):
    temp_k.append(np.bincount(label_k.reshape(10,-1)[i]).argmax())

new_list_k = []
for i in range(10):
    for j in range(100):
      new_list_k.append(temp_k[i])


score = rand_score(new_list_k, label_k)
score2 = mutual_info_score(new_list_k, label_k)
score3 = normalized_mutual_info_score(new_list_k, label_k)

print("rand_score: ",score)
print("mutual_info_score: ",score2)
print("normalized_mutual_info_score: ",score3)
```

```
rand_score:  0.8692972972972973
mutual_info_score:  1.0896206122566099
normalized_mutual_info_score:  0.49276537030432443
```

```
kmeans2 = cluster.KMeans(n_clusters = 10).fit(appx_kpca_R.reshape(-1,28*28))

temp_k2 = []
label_k2 = kmeans2.predict(appx_kpca_R.reshape(-1,28*28))


for i in range(10):
    temp_k2.append(np.bincount(label_k2.reshape(10,-1)[i]).argmax())


new_list_k2 = []
for i in range(10):
    for j in range(100):
      new_list_k2.append(temp_k2[i])


score = rand_score(new_list_k2, label_k2)
score2 = mutual_info_score(new_list_k2, label_k2)
score3 = normalized_mutual_info_score(new_list_k2, label_k2)

print("rand_score: ",score)
print("mutual_info_score: ",score2)
print("normalized_mutual_info_score: ",score3)
```

```
rand_score:  0.8726726726726727
mutual_info_score:  1.1030745339442753
normalized_mutual_info_score:  0.4981415061592738
```

오른쪽 코드는 PCA(10)을 적용시킨 후 Prediction 을 진행한 값과 실제 값에 대해서 rand_score 와 mutual_info_score 을 계산하였다. 그 결과 0.869 의 rand_score 를 보였고 normalized_mutual_info_score 는 0.492 가 나왔다. 주어진 수치만 놓고 보았을 때 K-means clustering 가 87%이상의 정확도를 보이며 각 클러스터를 비교할 수 있을 정도의 상호 의존도를 보임을 알 수 있다. Kernel PCA 을 적용한 결과를 보면 그 결과 0.873 의 rand_score 를 보였고 normalized_mutual_info_score 는 0.498 가 나왔다. 그냥 PCA 를 돌릴 때 보다 근소하게 높은 rand_score 값을 내었고 상호의존도 역시 0.5 미만의 점수를 기록하여 각 클러스터를 구분할 정도의 정보를 낼 수 있다는 것을 알 수 있다.

Q4. 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.

```
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics.pairwise import pairwise_distances_argmin_min
from sklearn.metrics import accuracy_score

centers, _ = pairwise_distances_argmin_min(kmeans.cluster_centers_,appx_pca_R)

num_predict = [[] for i in range(10)]
center_list = []

#print(centers)

for i in range(1000):
  for j in range(10):
    if label_k[i] == j:
      num_predict[j].append(i)

x_value = []
label = []

#for i in range(10):
#  center_list.append(num_predict[i][len(num_predict[i])//2])

for i in range(10):
  label.append(centers[i]//100)

for i in range(10):
  #x_value.append(appx_kpca_R[centers[i]//100][centers[i]%100])
  x_value.append(appx_pca_R[centers[i]])

#print(label)


knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_value,label)
prediction = knn.predict(new_X_Test)
score = classification_report(y_test,prediction)
print(score)
```

```
centers, _ = pairwise_distances_argmin_min(kmeans.cluster_centers_,appx_kpca_R)

num_predict = [[] for i in range(10)]
center_list = []

#print(centers)

for i in range(1000):
  for j in range(10):
    if label_k[i] == j:
      num_predict[j].append(i)

x_value = []
label = []

#for i in range(10):
#  center_list.append(num_predict[i][len(num_predict[i])//2])

for i in range(10):
  label.append(centers[i]//100)

for i in range(10):
  #x_value.append(appx_kpca_R[centers[i]//100][centers[i]%100])
  x_value.append(appx_kpca_R[centers[i]])


knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_value,label)
prediction = knn.predict(new_X_Test)
score = classification_report(y_test,prediction)
print(score)
```

	precision	recall	f1-score	support
0	0.88	0.80	0.84	980
1	0.71	0.95	0.81	1135
2	0.82	0.50	0.62	1032
3	0.00	0.00	0.00	1010
4	0.00	0.00	0.00	982
5	0.00	0.00	0.00	892
6	0.84	0.74	0.79	958
7	0.33	0.47	0.39	1028
8	0.28	0.70	0.40	974
9	0.32	0.72	0.45	1009
accuracy			0.50	10000
macro avg	0.42	0.49	0.43	10000
weighted avg	0.43	0.50	0.44	10000

	precision	recall	f1-score	support
0	0.75	0.91	0.82	980
1	0.67	0.96	0.79	1135
2	0.82	0.65	0.73	1032
3	0.55	0.70	0.62	1010
4	0.44	0.44	0.44	982
5	0.00	0.00	0.00	892
6	0.81	0.71	0.76	958
7	0.53	0.49	0.51	1028
8	0.56	0.52	0.54	974
9	0.38	0.55	0.45	1009
accuracy			0.60	10000
macro avg	0.55	0.59	0.56	10000
weighted avg	0.56	0.60	0.57	10000

	precision	recall	f1-score	support
0	0.85	0.64	0.73	980
1	0.51	0.89	0.65	1135
2	0.40	0.36	0.38	1032
3	0.00	0.00	0.00	1010
4	0.00	0.00	0.00	982
5	0.41	0.42	0.41	892
6	0.42	0.86	0.56	958
7	0.34	0.88	0.49	1028
8	0.43	0.35	0.39	974
9	0.00	0.00	0.00	1009
accuracy			0.45	10000

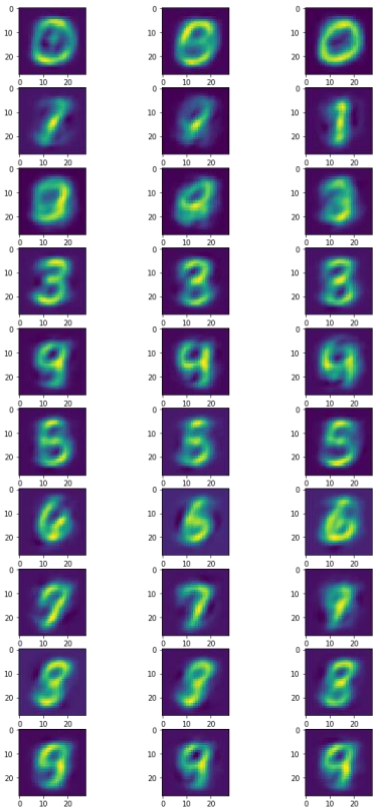
Accuracy from HW3

다음은 과제 3 번에서 했던 방식과 유사하게 Cluster 의 center 을 구한 다음에 이를 기준으로 1-NN classifier fitting 을 하고 fitting 된 classifier 을 실제 Mnist data set X_test 와 Y_test 에 적용하여서 실제로 PCA 와 Kernel PCA 을 사용한 분류 모델이 테스트 데이터에 대해서도 분류를 잘 할 수 있는지 측정하였다. 그 결과 그냥 PCA 를 썼을 경우에는 50%정도의 정확도를 보임을 알 수 있고 Kernel PCA 를 사용하였을 때는 이보다 약 10%정도 높은 60%의 정확도를 보임을 알 수 있다. Kernel PCA 를 사용하면 non-linear 한 decision boundary 를 표현하는 등 그냥 PCA 를 적용할 때보다 조금 더 정교한 모델을 제공하기 때문에 성능적으로 일반 PCA 보다 좋다고 분석할 수 있다. 또한 과제 3 번에서 PCA 를 적용하지 않고 kmeans clustering 과 1-nn classifier 을 적용해서 test set 에 적용한 45%의 정확도 보다 각각 약 5%, 15% 정도의 성능이 향상되었음을 알 수 있다. 이로 미루어 보았을 때 Mnist data set 분류할 때 PCA 기법을 적용하면 그냥 Clustering 을 했을 때 보다 조금 더 좋은 정확도를 보여준다는 것을 알 수 있다.

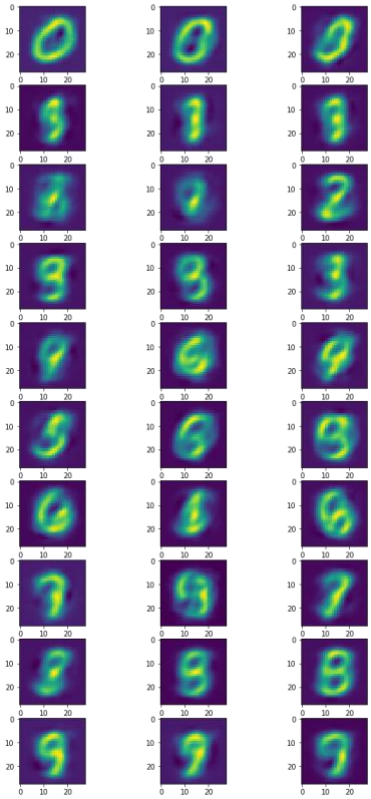
Q5. Visualize 3 correctly classified and 3 incorrectly classified images for each class. Explain your findings.

```
correct = [[]for i in range(10)]
incorrect = [[]for i in range(10)]


for i in range(10):
  c_cnt = 0
  i_cnt = 0

  for j in range(100):
    if np.array(label_k2).reshape(10,-1)[i][j] == np.array(new_list_k2).reshape(10,-1)[i][j] and c_cnt < 3:
      correct[i].append(appx_kpca_R[i*100+j])
      c_cnt += 1

    elif np.array(label_k2).reshape(10,-1)[i][j] != np.array(new_list_k2).reshape(10,-1)[i][j] and i_cnt < 3:
      incorrect[i].append(appx_kpca_R[i*100+j])
      i_cnt += 1

    else:
      pass


for i in range(10):
  for j in range(3):
    plt.imshow(correct[i][j].reshape(28,28))
    plt.title("correct image")
    plt.show()

for i in range(10):
  for j in range(3):
    plt.imshow(incorrect[i][j].reshape(28,28))
    plt.title("incorrect image")
```

correct image



incorrect image



다음은 PCA 를 통해서 복원한 image 를 대상으로 제대로 Classification 을 하는 경우엔 correct data set 으로, 제대로 Classification 을 하지 못하는 이미지는 incorrect data set 으로 분류해서 각 class 별로 3 개씩 출력한 결과이다. 실제로 correct 이미지의 경우에 비교적 clear 하게 숫자를 구분할 수 있는 형태를 보이지만 incorrect 이미지의 경우엔 숫자가 뭉개져서 나오거나 두 숫자 사이에서 구별하기 애매한 모양을 보임을 알 수 있다. (예를 들어 1 을 예측하는데 3 인지 1 인지 확실하게 구분할 수 없는 이미지가 나옴) 실제로 prediction 을 진행하여서 target label 을 출력해보면 우리가 원하는 대로 1 부터 9 까지의 classification 결과가 나오기를 바라지만 실제로 중복된 label 값이 나와서 다 깔끔하게 나누지는 못 하였다. 가장 대표적인 예시로 4 에 대한 classification 을 들 수 있는데 correct 데이터에서도 보다시피 출력된 이미지가 4 인지 9 인지 명확하게는 구분되지 않음을 알 수 있다. 따라서 이러한 경우에는 10 개의 축소된 feature 들로만 복원하지 않고 주성분의 수를 늘리거나 training 에 사용하는 data set 을 100 개씩만 활용하는 것이 아닌 더 많은 data set 을 이용한다면 조금 더 명확한 이미지가 나올 것이며 PCA 와 KPCA 의 정확도도 4 번에서 보았던 정확도 보다 더 높은 값을 보일 것이라고 예측할 수 있다.