CS302 Introduction to machine learning

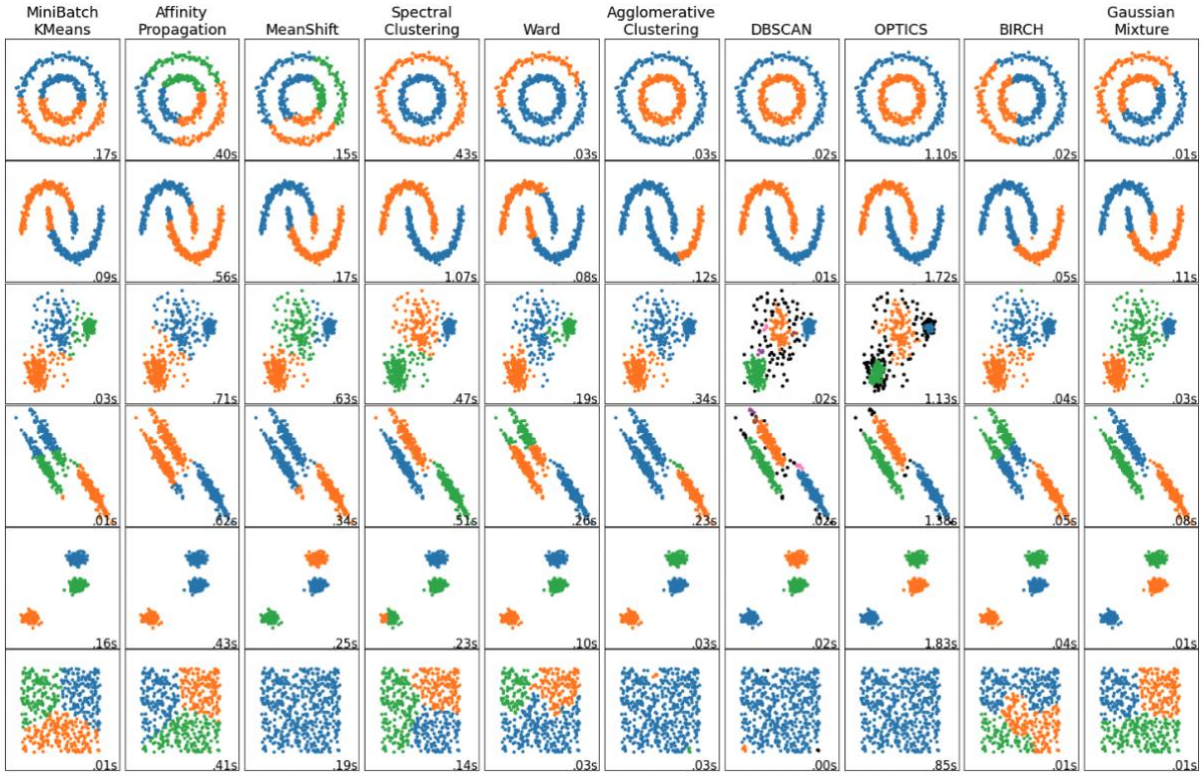
School of Undergraduate Studies, DGIST

201611161 정원균

Homework report 3.

2022. 05. 26

Q1. Run the code below and arrange the results.



Unsupervised Learning 중 다양한 Clustering 결과를 실행한 결과를 나타낸다.

Q2. Sample 100 images randomly for each class (total 1000 images) from the MNIST training data set.

```
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)
```

다음은 mnist 손글씨 데이터를 불러와서 X_train , Y_train, X_test, Y_test 를 설정해주었고, 각각의 dimension 을 맞춰주었다.

그 다음 X_train 과 y_train 이 mapping 되는 60000 개의 set 을 분류하여 (e.g. 0 으로 labeling 되면 0 번째 index 로 분류)한 다음 100 개의 샘플을 뽑는다. 이 과정에서 데이터의 편향을 막기 위해서 random.shuffle 메소드를 이용하여 각 index 마다 셔플링을 한 다음 100 개의 데이터를 추출하여 저장한다.

Q3. For 1000 images, perform Agglomerative clustering, k-means clustering, Gaussian mixture model, Spectral clustering. (i.e., k = 10)

```
from sklearn import cluster

agglo = cluster.AgglomerativeClustering(n_clusters = 10).fit(class_list.reshape(1000,-1))
spectral = cluster.SpectralClustering(n_clusters = 10, eigen_solver = 'arpack',affinity='nearest_neighbors').fit(class_list.reshape(1000,-1))
GMM = mixture.GaussianMixture(n_components = 10, covariance_type = 'full').fit(class_list.reshape(1000,-1))
kmeans = cluster.KMeans(n_clusters = 10).fit(class_list.reshape(1000,-1))
```

1000 개의 image 들 (100 x 10)에 대해서 Agglomerative clustering, k-means clustering, Gaussian mixture model, Spectral clustering 을 10 개의 cluster 로 clustering 을 하였다.

Q4. Based on the clustering results and the labels we know, compute "Rand index" and "mutual information-based score". Explain your findings.

4-1) Agglomerative Clustering

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

temp_a = []
label_a = agglo.fit_predict(class_list.reshape(1000,-1))


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


new_list_a = []
for i in range(10):
    for j in range(100):
      new_list_a.append(temp_a[i])


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

```
rand_score:  0.8893373373373373
mutual_info_score:  1.3810659822448432
normalized_mutual_info_score:  0.6432219475045975
```

클러스터링의 성능을 평가하는 대표적인 기준으로 Rand_index 와 Mutual information score 이 있는데 Rand_index 는 비교할 두 데이터의 순서쌍을 비교하여서 모든 데이터에 대해 일치하는 데이터 쌍의 갯수를 비율로 나타내며 1 에 가까울 수록 좋은 성능의 클러스터링이라는 것을 알 수 있다. 하지만 cluster 의 수가 증가할 수록 Rand_index 값이 증가하기 때문에 실제 정확도에 비해서 더 높은 정확도가 나오는 경향성이 있다.

또한 Mutual information score 는 두 확률 변수의 의존도를 수치화한 점수이다. 이는 Correlation 과 관계가 있는데 Correlation 값이 높을 수록 두 확률 변수가 서로 의존도가 높고 0 에 가까울 수록 두 확률변수가 독립적이라는 것을 의미한다. 역시 Cluster 수가 증가하면 확률변수의 상호의존 정도가 낮아지기 때문에 실제보다 조금 더 낮은 값이 나올 수 있다.

다음은 agglomerative clustering 을 진행하고 나온 image 와 label 을 추출하여 Rand_index 와 Mutual information score 을 구해보았다.

약 0.889 의 rand_index 와 0.643 normalized Mutual information score 을 보임을 알 수 있다. 주어진 수치만 놓고 보았을 때 agglomerative clustering 이 좋은 성능을 보이고 각 클러스터를 비교할 수 있을 정도의 상호 의존도를 보임을 알 수 있다.

4-2) Spectral Clustering

```
temp_s = []
label_s = spectral.fit_predict(class_list.reshape(1000,-1))

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

new_list_s = []
for i in range(10):
    for j in range(100):
      new_list_s.append(temp_s[i])


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

```
rand_score:  0.8753693693693694
mutual_info_score:  1.2417560181051674
normalized_mutual_info_score:  0.6256403779104138
```

다음은 Spectral clustering 을 진행하고 나온 image 와 label 을 추출하여 Rand_index 와 Mutual information score 을 구해보았다.

약 0.875 의 rand_index 와 0.625 normalized Mutual information score 을 보임을 알 수 있다. 주어진 수치만 놓고 보았을 때 Spectral clustering 이 좋은 성능을 보이고 각 클러스터를 비교할 수 있을 정도의 상호 의존도를 보임을 알 수 있다.

4-3) GMM

다음은 Gaussian Mixture model 을 진행하고 나온 image 와 label 을 추출하여 Rand_index 와 Mutual information score 을 구해보았다.

```
temp_g = []
label_g = GMM.fit_predict(class_list.reshape(1000,-1))

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

new_list_g = []
for i in range(10):
    for j in range(100):
      new_list_g.append(temp_g[i])


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

```
rand_score:  0.8666806806806807
mutual_info_score:  1.018635660780017
normalized_mutual_info_score:  0.47637057051509507
```

약 0.867 의 rand_index 와 0.476 normalized Mutual information score 을 보임을 알 수 있다. 주어진 수치만 놓고 보았을 때 Gaussian Mixture model 이 좋은 성능을 보이고 각 클러스터를 비교할 수 있을 정도의 상호 의존도를 보임을 알 수 있다. 여기서 다른 모델들 보다 상호 의존도가 낮아서 좀 더 클러스터간의 비교하기에 용이한 것을 알 수 있다.

4-4) K-means Clustering

```
temp_k = []
label_k = kmeans.fit_predict(class_list.reshape(1000,-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:  0.8622042042042042
mutual_info_score:  1.0716688762945468
normalized_mutual_info_score:  0.50071421447699
```

다음은 K-means clustering 을 진행하고 나온 image 와 label 을 추출하여 Rand_index 와 Mutual information score 을 구해보았다.

약 0.862 의 rand_index 와 0.501 normalized Mutual information score 을 보임을 알 수 있다. 주어진 수치만 놓고 보았을 때 K-means clustering 이 좋은 성능을 보이고 각 클러스터를 비교할 수 있을 정도의 상호 의존도를 보임을 알 수 있다. 역시 앞의 두 모델들 보다 상호 의존도가 낮아서 좀 더 클러스터간의 비교하기에 용이한 것을 알 수 있다.

Cluster type	Agglomerative	Spectral	GMM	K-means
RI	0.889	0.875	0.867	0.862
MIS	1.381	1.241	1.019	1.072
NMIS	0.643	0.625	0.476	0.501

Q5. 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.

4-1) Agglomerative Clustering

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

for i in range(1000):
  for j in range(10):
    if label_a[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(center_list[i]//100)

for i in range(10):
  x_value.append(class_list.reshape(1000,-1)[center_list[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       0.86      0.53      0.66       980
           1       0.41      0.49      0.45      1135
           2       0.73      0.15      0.25      1032
           3       0.31      0.79      0.44      1010
           4       0.00      0.00      0.00       982
           5       0.00      0.00      0.00       892
           6       0.75      0.59      0.66       958
           7       0.25      0.86      0.39      1028
           8       0.51      0.50      0.51       974
           9       0.00      0.00      0.00      1009

    accuracy                           0.40     10000
```

다음은 Agglomerative Clustering 을 처리한 다음에 0~9 까지 각 cluster 의 center 값을 찾아서 label 과 train image 값을 리스트에 넣어서 처리하였다. 그 다음 1-NN classifier 을 이용하여 center 값들에 대해 fitting 을 진행하였다. 학습이 완료된 1-NN classifier 로 실제 Test image prediction 을 하였고 y_test 값과 Prediction 한 값의 정확도를 구하였다.

그 결과 총 40%의 정확도가 나옴을 알 수 있는데, 총 6 만개의 data 값들 중에 각 label 당 100 개씩의 data 만 Sampling 하여 clustering 을 진행한 후 classifier 로 predict 를 진행하였기 때문에 적당한 정확도를 보이지만 더 많은 데이터를 sampling 하여 clustering 한 후 classifier 로 예측을 진행하면 더 좋은 정확도가 나올 것으로 예상할 수 있다.

4-2) Spectral Clustering

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

for i in range(1000):
  for j in range(10):
    if label_s[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(center_list[i]//100)

for i in range(10):
  x_value.append(class_list.reshape(1000,-1)[center_list[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       0.74      0.73      0.73       980
           1       0.36      0.94      0.52      1135
           2       0.72      0.38      0.49      1032
           3       0.00      0.00      0.00      1010
           4       0.34      0.43      0.38       982
           5       0.36      0.51      0.42       892
           6       0.68      0.65      0.66       958
           7       0.51      0.48      0.49      1028
           8       0.35      0.41      0.38       974
           9       0.00      0.00      0.00      1009

    accuracy                           0.46     10000
```

다음은 Spectral Clustering 을 처리한 다음에 0~9 까지 각 cluster 의 center 값을 찾아서 label 과 train image 값을 리스트에 넣어서 처리하였다. 그 다음 1-NN classifier 을 이용하여 center 값들에 대해 fitting 을 진행하였다. 학습이 완료된 1-NN classifier 로 실제 Test image prediction 을 하였고 y_test 값과 Prediction 한 값의 정확도를 구하였다.

그 결과 총 46%의 정확도가 나옴을 알 수 있는데, 총 6 만개의 data 값들 중에 각 label 당 100 개씩의 data 만 Sampling 하여 clustering 을 진행한 후 classifier 로 predict 를 진행하였기 때문에 적당한 정확도를 보이지만 더 많은 데이터를 sampling 하여 clustering 한 후 classifier 로 예측을 진행하면 더 좋은 정확도가 나올 것으로 예상할 수 있다.

4-3) GMM

다음은 Gaussian Mixture Model 을 처리한 다음에 0~9 까지 각 cluster 의 center 값을 찾아서 label 과 train image 값을 리스트에 넣어서 처리하였다. 그 다음 1-NN classifier 을 이용하여

center 값들에 대해 fitting 을 진행하였다. 학습이 완료된 1-NN classifier 로 실제 Test image prediction 을 하였고 y_test 값과 Prediction 한 값의 정확도를 구하였다.

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

for i in range(1000):
  for j in range(10):
    if label_g[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(center_list[i]//100)

for i in range(10):
  x_value.append(class_list.reshape(1000,-1)[center_list[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       0.60      0.77      0.67       980
           1       0.43      0.90      0.58      1135
           2       0.00      0.00      0.00      1032
           3       0.54      0.19      0.28      1010
           4       0.00      0.00      0.00       982
           5       0.39      0.28      0.32       892
           6       0.63      0.73      0.68       958
           7       0.63      0.66      0.64      1028
           8       0.37      0.36      0.36       974
           9       0.26      0.59      0.36      1009

    accuracy                           0.45     10000
```

그 결과 총 45%의 정확도가 나옴을 알 수 있는데, 총 6 만개의 data 값들 중에 각 label 당 100 개씩의 data 만 Sampling 하여 clustering 을 진행한 후 classifier 로 predict 를 진행하였기 때문에 적당한 정확도를 보이지만 더 많은 데이터를 sampling 하여 clustering 한 후 classifier 로 예측을 진행하면 더 좋은 정확도가 나올 것으로 예상할 수 있다.

4-4) K-means Clustering

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

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

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(center_list[i]//100)

for i in range(10):
  x_value.append(class_list.reshape(1000,-1)[center_list[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       0.85      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
```

다음은 K-means Clustering 을 처리한 다음에 0~9 까지 각 cluster 의 center 값을 찾아서 label 과 train image 값을 리스트에 넣어서 처리하였다. 그 다음 1-NN classifier 을 이용하여 center 값들에 대해 fitting 을 진행하였다. 학습이 완료된 1-NN classifier 로 실제 Test image prediction 을 하였고 y_test 값과 Prediction 한 값의 정확도를 구하였다.

그 결과 총 45%의 정확도가 나옴을 알 수 있는데, 총 6 만개의 data 값들 중에 각 label 당 100 개씩의 data 만 Sampling 하여 clustering 을 진행한 후 classifier 로 predict 를 진행하였기 때문에 적당한 정확도를 보이지만 더 많은 데이터를 sampling 하여 clustering 한 후 classifier 로 예측을 진행하면 더 좋은 정확도가 나올 것으로 예상할 수 있다.

Clustering type	Agglomerative	Spectral	GMM	K-means
Accuracy	40%	46%	45%	45%

최종 결과를 본다면 가장 Naïve 한 Agglomerative Clustering 이 가장 낮은 정확도를 보이고 나머지 3 개의 Clustering 은 비슷하게 조금 더 높은 정확도를 보임을 알 수 있다.