수치해석

Final Term Project # Pattern recognition

컴퓨터 소프트웨어 학부 2016025469 서건식

1. Generate 5 Classes data using random number generation(Gaussian)

```
def generateSample(m_x,sig_x,m_y,sig_y,m_z,sig_z,size,index,df_input):
    X = np.random.normal(m_x, sig_x, size)
    Y = np.random.normal(m_y, sig_y, size)
```

Sample을 생성하는 함수에서는 각 x,y,z 좌표에 대한 평균과 분산을 입력하고,
사이즈를 넣어 생성합니다.

```
df = pd.DataFrame(columns=['x','y','z'])

generateSample(0, 1, 0, 1, 1, 1,300,0,df)
generateSample(0, 1.5, 0, 1.5, 5.5, 1,300,300,df)
generateSample(3, 1.3, 3, 1.3, 7, 1.5,300,600,df)
generateSample(7, 1.5, 7, 1.5, 5.5, 1,300,900,df)
generateSample(7, 1, 7, 1, 1, 1,300,1200,df)
```

	x	y	z
0	-1.825084	1.872317	0.679500
1	0.066110	-0.672062	0.514955
2	1.174054	-1.112424	0.471546
3	-0.711255	-0.210216	0.753503
4	-2.403707	0.163719	1.210036
...	...	...	...
1495	5.804897	3.684292	0.009889
1496	6.689533	6.694803	2.127886
1497	6.671048	7.008707	1.445336
1498	6.941975	5.115183	0.879047
1499	7.246975	8.017782	1.873361

1500 rows × 3 columns

이렇게 총 1500개의 Sample을 생성합니다.

생성한 각 클래스의 평균과 분산은 다음과 같습니다.

	X_mu	X_sig	Y_mu	Y_sig	Z_mu	Z_sig
Class1	0	1	0	1	1	1
Class2	0	1.5	0	1.5	5.5	1
Class3	3	1.3	3	1.3	7	1.5
Class4	7	1.5	7	1.5	5.5	1
Class5	7	1	7	1	1	1

2. K-means Clustering for 3D vector data

```
Class1's cluster =  3
Class2's cluster =  1
Class3's cluster =  4
Class4's cluster =  2
Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```

K – Means 클러스터링은 5개의 Cluster로 설정한 후 시행하였으며, 각 클래스마다 위와 같이 Mapping이 되었습니다.
Class1은 3 Cluster, Class2는 1 Cluster … Class5는 0 Cluster입니다.

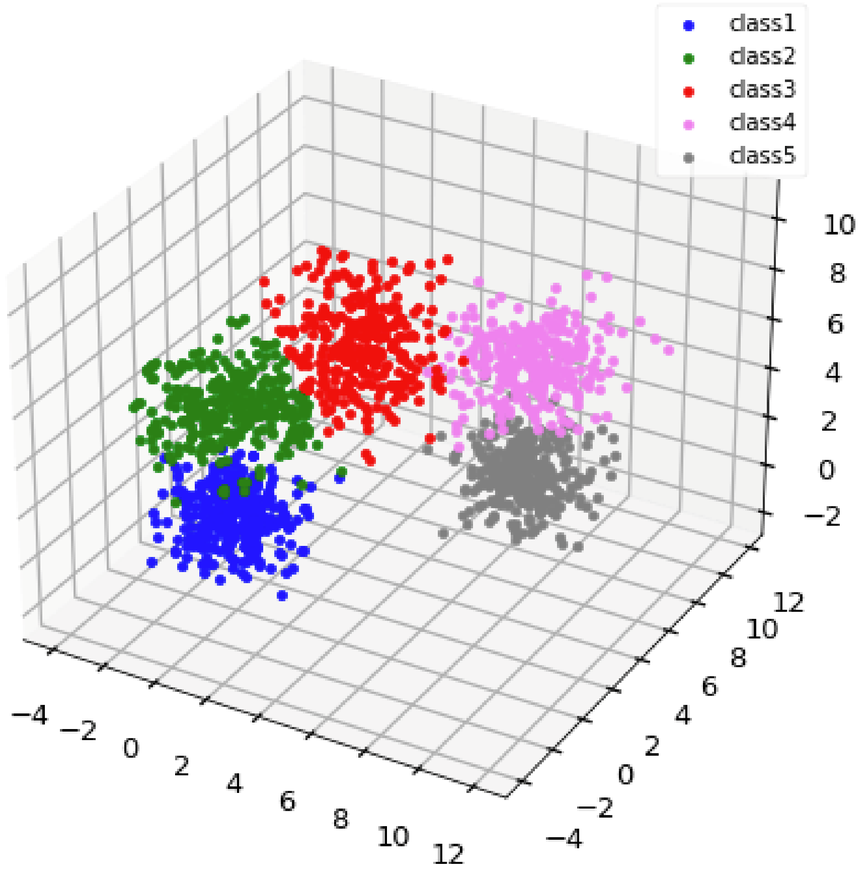
3	-0.711255	-0.210216	0.753503	3
4	-2.403707	0.163719	1.210036	3
5	-0.479459	-0.330108	0.800648	3
6	-0.404025	2.018095	-1.030411	3
7	-0.158267	-0.219414	3.258403	1
8	-0.078070	0.864184	0.701620	3
9	-2.644278	1.956988	-0.041485	3
10	-0.291174	0.092523	0.113450	3
11	-0.351993	0.212534	2.319065	3
12	-0.043236	-1.496154	1.591882	3
13	1.255752	0.670282	0.831593	3
815	1.046765	3.372443	7.161048	4
816	2.802406	2.514766	5.768427	4
817	2.763203	3.498933	5.974652	4
818	0.706907	1.895536	5.750568	1
819	1.842526	4.243500	7.103503	4

Modeling을 진행한 후, Class 별로 생성한 데이터가 각각 옆 군집으로 이동한 것을 볼 수 있습니다. 이는 각 Class 마다 일부러 겹치는 구간이 있도록 설정한 것이며, 대략적으로 5-10 개의 Sample data가 겹치게 됩니다.

예를 들어 7번째의 Data는 원래는 Class1의 데이터로 Cluster가 3으로 분류되어야 하나 그 옆인 Class2의 Cluster인 1로 분류 됐음을 확인할 수 있습니다.

밑쪽의 818번째 데이터 또한 Class3에서 Generation 된 샘플이지만 Class2의 Cluster인 1로 Clustering 되었습니다.

2. K-means Clustering for 3D vector data




```
Class1's cluster =  3
Class2's cluster =  1
Class3's cluster =  4
Class4's cluster =  2
Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```

K-Means Clustering을 한 후 Sample Data들의 분포입니다.

각 클래스에 해당하는 Cluster Number와 Mapping을 시켜놨고, 분석에 용이하도록 추상화 했습니다.
데이터의 분포를 시각적으로 봤을 때, Class 간 경계선에 데이터가 겹쳐 있는 모습을 볼 수 있습니다.

2. K-means Clustering for 3D vector data

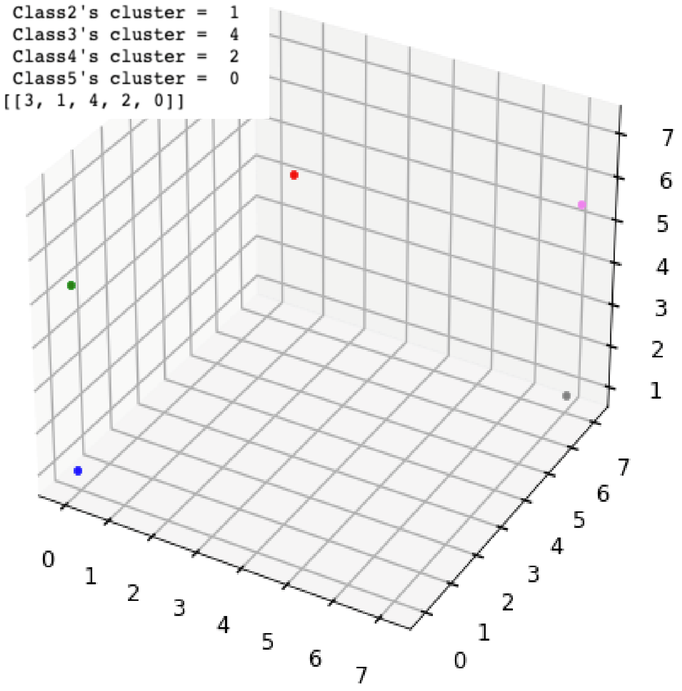
```
print(meanDistance())
```

```
4.483863274908919
```

```
kmeans.cluster_centers_
```

```
array([[ 7.07041413,  6.91288739,  1.03079774],
       [-0.03709025, -0.04751094,  5.34609844],
       [ 7.13698603,  7.1115972 ,  5.50976106],
       [-0.09090603, -0.00802735,  0.97640756],
       [ 3.08345669,  2.97471418,  7.18233194]])
```

```
Class1's cluster =  3
Class2's cluster =  1
Class3's cluster =  4
Class4's cluster =  2
Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```



K-Means Clustering을 한 후 각 Cluster의 Mean Vector를 구해 표시한 그림입니다.

Array의 순서가 섞여있는데, Array의 순서는 Cluster의 순서로 나열되어 출력되므로 실제로는
Class1 = array[3], Class2 = array[1] ... Class5 = array[0] 이라고 생각하면 됩니다. 따라서 순서는

Class1's Mean Vector: [-0.09090603, -0.00802735, 0.97640756]

Class2's Mean Vector: [-0.03709025, -0.04751094, 5.34609844]

Class3's Mean Vector: [3.08345669, 2.97471418, 7.18233194]

Class4's Mean Vector: [7.13698603, 7.1115972 , 5.50976106]

Class5's Mean Vector: [7.07041413, 6.91288739, 1.03079774]

입니다. meanDistance는 각 Cluster마다 옆 Cluster 들과의 거리를 더해 평균을 낸 것으로, 4.4838
이 나왔습니다. 각 Cluster Mean Vector 간 거리의 평균이 4.4838이라는 것입니다.

Class에 속하지 않는 다른 분포를 가지는 Test Data를 판별할 때 사용할 예정입니다.

3. Testing with new vectors

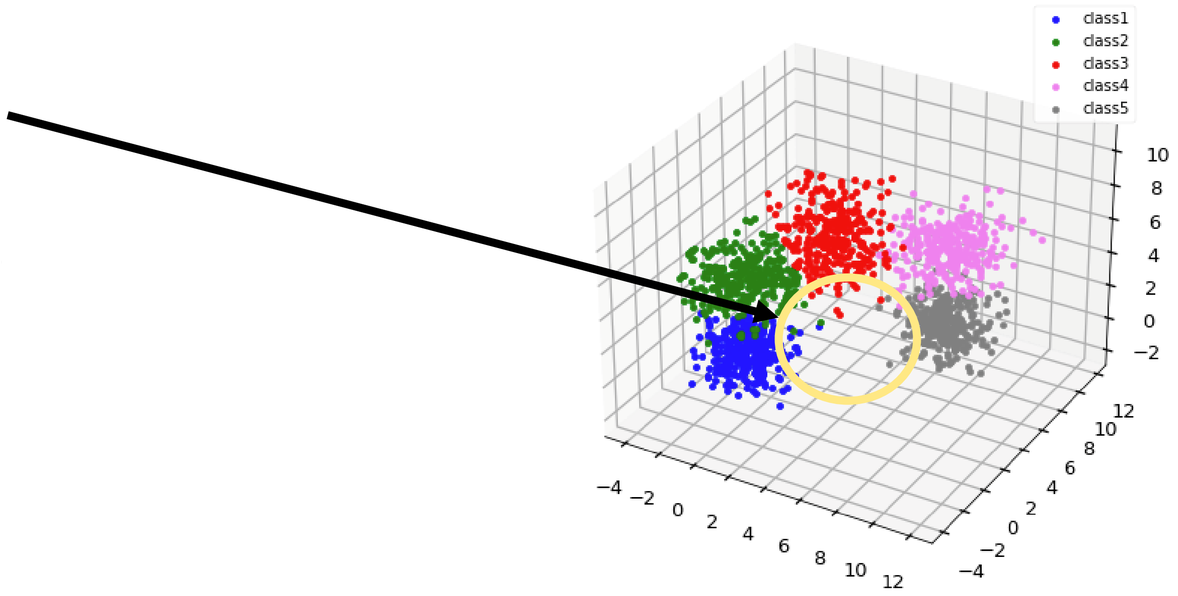
```
df_sample = pd.DataFrame(columns=['x','y','z'])

generateSample(0, 1, 0, 1, 1, 1,100,0,df_sample)
generateSample(0, 1.5, 0, 1.5, 5.5, 1,100,100,df_sample)
generateSample(3, 1.3, 3, 1.3, 7, 1.5,100,200,df_sample)
generateSample(7, 1.5, 7, 1.5, 5.5, 1,100,300,df_sample)
generateSample(7, 1, 7, 1, 1, 1,100,400,df_sample)

#분포가 다른 Sample 100개
generateSample(3,1,3,1,1,1,100,500,df_sample)
```

Random Data generation을 진행할 때, 같은 Gaussian 분포를 가지는 sample을 100개씩 총 5클래스에 대하여 500개를 뽑습니다.

다른 분포를 가지는 sample 또한 뽑는데, Class1, 3, 5의 사이에 위치하는 분포에 샘플을 생성합니다.

Classify는 각 Cluster의 Mean Vector와, 데이터의 유클라디안 거리를 측정하여 가장 가까운 Mean Vector가 속한 Cluster로 Classify하도록 알고리즘을 만들었습니다.

	X_mu	X_sig	Y_mu	Y_sig	Z_mu	Z_sig
Class1	0	1	0	1	1	1
Class2	0	1.5	0	1.5	5.5	1
Class3	3	1.3	3	1.3	7	1.5
Class4	7	1.5	7	1.5	5.5	1
Class5	7	1	7	1	1	1
Test Other Distribution	3	1	3	1	1	1



3. Testing with new vectors – Class1, Class2

```
Class1's cluster =  3
Class2's cluster =  1
Class3's cluster =  4
Class4's cluster =  2
Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```

Class1

```
#정확도를 계산하기 위해 인덱스를 넣는다
# 0- 100 = class1 / 100 - 200 = class2 ... 이런 식으로 df_sample에 들어가 있다.
# 따라서 인덱스를 구분해주고, 추측한 결과가 원래 클래스와 맞는지를 검사한다.

cluster1, accuracy1= class_recognition(0)
print("분류된 Cluster:\n",cluster1)
print(accuracy1,"%의 정확도")
```

```
100
분류된 Cluster:
 [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3]
96.0 %의 정확도
```

Class2

```
cluster2, accuracy2= class_recognition(100)
print("분류된 Cluster:\n",cluster2)
print(accuracy2,"%의 정확도")
```

```
100
분류된 Cluster:
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1]
96.0 %의 정확도
```

Class1과 Class2과 같은 분포로 생성된 Test 데이터들은 각각 Cluster number가 3, 1로 분류되어야 Recognition이 잘 수행되었다고 할 수 있습니다. 전체 100개의 Sample중 Class1과 Class2는 약 96%의 정확도를 보였습니다. 다르게 분류된 데이터가 어느 Cluster로 분류되었는지 살펴보면, Class1은 인접한 Cluster가 Class2 이므로 1로 분류되었고, Class2는 Class1과 Class3이므로 3과 4로 분류됨을 볼 수 있습니다. 이 데이터들은 Cluster간 경계에 있는 데이터입니다.

같은 분포의 샘플임에도 다르게 분류되는 이유는 Class1의 데이터 중 Class2로 분류된 데이터는 Class2의 Mean Vector와 그 데이터가 더 가깝다는 것 때문입니다. 이 현상이 나타나는 이유는 각 Class 당 500개의 샘플을 생성하고 K-means Clustering으로 모델을 제작했을 때 몇 샘플이 다른 군집으로 이동하게 되는 현상과 같은 이유입니다. Test Sample은 만약 Class1과 같은 분포로 형성한 샘플이라고 하더라도 Class1은 K-means Clustering으로 분류되어 나온 모델임으로 끝 쪽의 데이터는 커버하지 못할 수 있습니다.

즉, 정확도라는 것은 단지 제가 같은 분포의 샘플이 같은 분포로 학습시킨 Clustering으로 Classify 되는지에 대한 정확도를 테스트한 것이며, 해당 Cluster에 인접한 다른 Cluster로 분류되었다 하더라도 그것이 잘못 분류되었다고는 할 수 없습니다.

3. Testing with new vectors – Class3, Class4

```
Class1's cluster =  3
Class2's cluster =  1
Class3's cluster =  4
Class4's cluster =  2
Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```

Class3

```
cluster3, accuracy3= class_recognition(200)
print("분류된 Cluster:\n",cluster3)
print(accuracy3,"%의 정확도")
```

```
100
분류된 Cluster:
 [4, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
94.0 %의 정확도
```

Class4

```
cluster4 ,accuracy4= class_recognition(300)
print("분류된 Cluster:\n",cluster4)
print(accuracy4,"%의 정확도")
```

```
100
분류된 Cluster:
 [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2]
97.0 %의 정확도
```

Class3과 Class4과 같은 분포로 생성된 Test 데이터들은 각각 Cluster number가 4, 2로 분류되어야 Recognition이 잘 수행되었다고 할 수 있습니다.

Class3과 인접한 Class2, Class4의 Cluster number인 1과 2로 각각 다르게 분류되며, Class4도 마찬가지로 인접한 Cluster number인 4와 0 으로 각각 다르게 분류됨을 확인할 수 있습니다.

3. Testing with new vectors – Class5

```
 Class1's cluster =  3
 Class2's cluster =  1
 Class3's cluster =  4
 Class4's cluster =  2
 Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```

Class5

```
cluster5, accuracy5= class_recognition(400)
print("분류된 Cluster:\n",cluster5)
print(accuracy5,"%의 정확도")
```

```
100
분류된 Cluster:
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
100.0 %의 정확도
```

```
cluster5, accuracy5= class_recognition(400)
print("분류된 Cluster:\n",cluster5)
print(accuracy5,"%의 정확도")
```

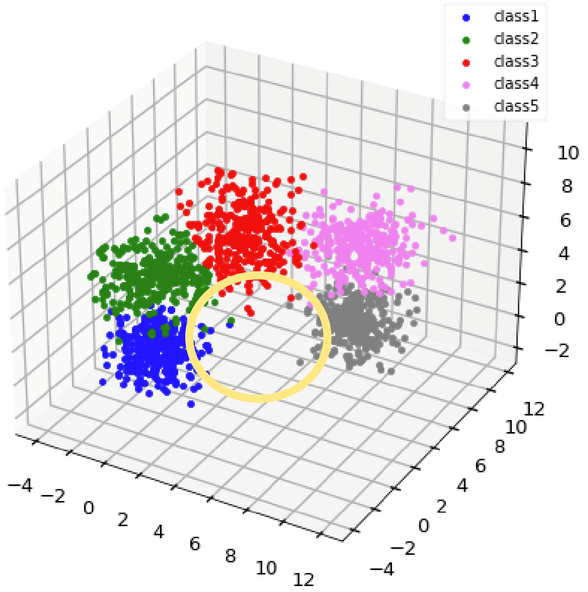
```
100
분류된 Cluster:
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
99.0 %의 정확도
```

Class5와 같은 분포로 생성된 Test 데이터들은 Cluster number가 0으로 분류되어야 Recognition이 잘 수행되었다고 할 수 있습니다.

위 경우는 모두 0으로 분류가 되었는데, 이렇게 100%의 정확도가 나왔다고 하더라도 이것은 생성된 Sample 데이터가 Class5의 Mean Vector 쪽으로 더 치우치게 생성이 되었기 때문이며 한번 더 Sample data를 추출해 본다면 다른 결과가 나옵니다.

3. Testing with new vectors – Different Distribution

```
Class1's cluster =  3
Class2's cluster =  1
Class3's cluster =  4
Class4's cluster =  2
Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```



	X_mu	X_sig	Y_mu	Y_sig	Z_mu	Z_sig
Test Other Distribution	3	1	3	1	1	1
Class1	0	1	0	1	1	1
Class3	3	1.3	3	1.3	7	1.5
Class5	7	1	7	1	1	1

다른 분포로 생성할 샘플은, Class1과 3, 5에 걸치도록 생성했습니다.

이 샘플 데이터를 생성한 목적은, 샘플 데이터 분포에서 중앙 쪽에 위치한 데이터들은 해당 Cluster로 분류하지 않으면서, 경계에 존재하는 데이터들은 각각 Class1, 3, 5로 분류하도록 Model을 수정하기 위함 입니다.

그 방법으로 조건을 하나 더 추가했습니다. 유클리드 거리를 측정하여 가장 가까운 Mean Vector에 해당하는 Cluster를 골랐을 때, 그 때 측정한 거리가 위에서 구했던,

```
print(meanDistance())
```

```
4.483863274908919
```

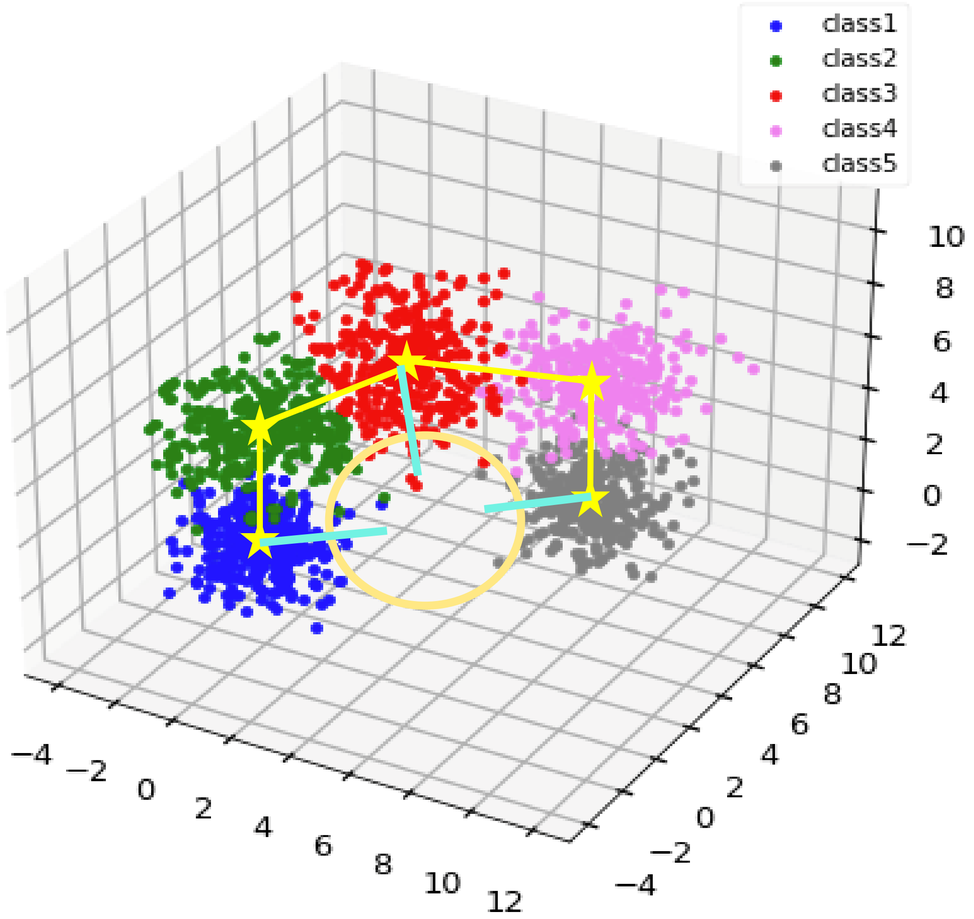
의 K배보다 짧은 경우에만 분류를 시키는 방법을 고려합니다.

3. Testing with new vectors – Different Distribution

```
Class1's cluster =  3
Class2's cluster =  1
Class3's cluster =  4
Class4's cluster =  2
Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```

```
print(meanDistance())
```

```
4.483863274908919
```



이웃한 Cluster간 거리의 평균은 4.48로 나왔으며, 2.24정도가 각 클러스터의 평균 반지름이라고 할 수 있습니다.

최대로 수용할 수 있는 거리가 meanDistance 그대로 4.48이라면, 중앙에 있는 데이터도 모두 분류를 해 버릴 것입니다. 따라서 적절한 값을 찾기 위해 그림상으로는 파란 형광색 선(거리)를 조금씩 줄여가며 K값을 구하고, 경계선의 데이터만을 분류할 수 있도록 할 것입니다.

정리하자면 Test Sample들을 S라고 하고 그 안의 데이터들을 S[i1,i2 ..., I100]라고 하고, 이웃한 Cluster 간 거리의 평균을 M이라고 하겠습니다.

S[in]과 각 Cluster의 Mean Vector의 거리를 구했을 때 가장 가까운 Mean Vector와의 거리를 D라고 하면

D > M/K인 데이터 S[in]을 버린다고 했을 때 중앙의 데이터만을 제외하고 경계에 근접한 데이터를 취하는 적절한 K를 찾는 것입니다.

3. Testing with new vectors – Different Distribution

```
Class1's cluster =  3
Class2's cluster =  1
Class3's cluster =  4
Class4's cluster =  2
Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```

```
print(meanDistance())
```

```
4.483863274908919
```

```
cluster6, accuracy6 = class_recognition(500)
print("분류된 Cluster:\n",cluster6)
```

```
100
분류된 Cluster:
 [4, 3, 3, 3, 0, 3, 3, 3, 3, 0, 0, 3, 0, 0, 3, 3, 3, 0, 0, 3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3,
3, 0, 3, 3, 0, 3, 0, 3, 3, 3, 0, 0, 3, 0, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 0, 3, 0, 0, 3, 3, 3, 0, 0, 3, 3, 0, 0, 3, 3,
3, 0, 0, 3, 3, 3, 3, 0, 0, 3, 0, 3, 3, 0, 3, 4, 3, 3, 3, 0, 3, 3, 3]
```

D > M/1

```
elif(expected > meanDistance()/1):
    cluster_num = -1
```

이웃한 클러스터간 거리의 평균을 그대로 사용했을 때 입니다. 분류된 Cluster를 보면 4,3,0으로 각각 Class 3, 1, 5입니다. 이를 보면 Sample 데이터의 모든 데이터가 분류가 되어버렸으므로 OverFitting 된 상황입니다. 가운데에 있는 데이터까지 모두 분류해 버린 경우입니다.

```
cluster6, accuracy6 = class_recognition(500)
print("분류된 Cluster:\n",cluster6)
```

```
100
분류된 Cluster:
 [-1, -1, 4, -1, -1, -1, -1, -1, 3, -1, -1, -1, -1, -1, -1, -1, 3, -1, -1, -1, -1, -1, 3, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 3, -1, -1, -1, -1, -1, -1, -1, -
1, 3, -1, -1, -1, 3, -1, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 4, -1, 3, -1, -1, 3, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, 3, -1]
```

D > M/1.4

```
elif(expected > meanDistance()/1.4):
    cluster_num = -1
```

-1은 분류가 안된 것입니다. 여기서는 3,4 Cluster로 분류가 된 것으로 보이는데, 이는 Class1과 Class3의 경계에 위치한 데이터가 잘 분류된 것입니다. 하지만 Class5에 대한 경계값은 제대로 분류되지 않은 것으로 보입니다. 따라서 길이를 조금 더 길게 설정해줄 필요가 있습니다.

3. Testing with new vectors – Different Distribution

```
Class1's cluster =  3
Class2's cluster =  1
Class3's cluster =  4
Class4's cluster =  2
Class5's cluster =  0
[[3, 1, 4, 2, 0]]
```

```
print(meanDistance())
```

```
4.483863274908919
```

```
cluster6, accuracy6 = class_recognition(500)
print("분류된 Cluster:\n",cluster6)
```

```
100
분류된 Cluster:
 [-1, -1, -1, 3, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, 3, -1, -1, -1, 3, -1, 3, -1, 3, -1, 3, -
1, -1, -1, -1, -1, -1, 3, 3, -1, -1, -1, -1, -1, -1, 3, -1, -1, -1, 4, -1, -1, 3, 0, -1, 3, 3, 3, -1, -1, -1, -1,
3, 3, -1, -1, -1, -1, -1, 3, -1, 0, -1, -1, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 4, -1, -1, -1,
-1, -1, 3, -1, -1, -1, -1]
```

D > M/1.3

```
elif(expected > meanDistance()/1.3):
    cluster_num = -1
```

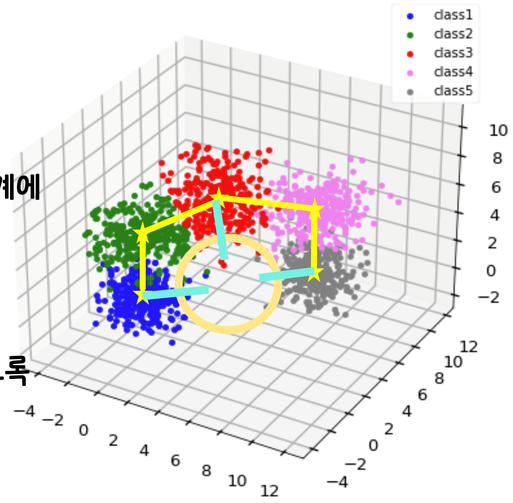
Maximum distance를 조금 더 길게 한 결과, Cluster가 0,3,4 / 즉 Class 1,3,5와 걸쳐있는, 즉 경계선에 해당하는 S[in] 데이터들은 각 Cluster로 구분되면서 가운데에 해당하는, 즉 Maximum distance (M/1.3)보다 더 먼 곳에 위치한 중앙 쪽에 분포하는 S[in] 데이터는 분류되지 않았음을 확인할 수 있었습니다.

따라서 제가 만든 Model의 적절한 Maximum Distance 값은 M/1.3으로, 그 값은 3.449125592307692 였습니다. 물론, 샘플을 다시 생성하고 K-Means Clustering을 할 때 마다 달라지지만 보고서를 작성할 때 생성된 Sample의 경우에는 그렇습니다.

4. 결론

5개의 Gaussian 분포를 따르는 각각 300개의 데이터를 어느정도 경계선에 데이터가 겹치도록 1500개를 생성하고, 그 Data set을 K-means Clustering으로 학습하여 Model을 만들었다. 그 결과 처음에 만들었던 데이터에서 어느정도 군집의 이동이 발생했다.

위 모델이 Classify를 잘 하는지 Test하기 위해서 초반에 만든 분포와 같은 5개의 Sample data set을 100개씩 500개 추출하고, 해당 분포와 다른 Gaussian 분포를 가지는 Sample Data set을 100개 추출한다.



같은 분포를 가지는 5개의 Data set은 대체로 높은 정확도를 보였으나 이웃 Cluster로 분류되는 경우도 있었다. 그 이유는 처음부터 Data set을 어느정도 겹치게 생성시켰고, 이 Data set을 K-means Clustering으로 학습시킨다면 원래 Data set의 경계에 있던 값이 다른 군집으로 이동하기 때문이다. Test Data set은 경계에 있는 데이터 또한 생성하기 때문에, 이 데이터가 다른 군집으로 Classify 되는 것이다.

다른 Gaussian 분포를 가지는 100개의 Sample은 그 경계선을 각각 Class1, 3, 5에 걸치도록 하였다. 그 이유는 해당 Test Sample의 중앙값 까지 분류하는 OverFitting을 막기 위해서, 가장 가까운 Mean Vector에 해당하는 Cluster를 찾았더라도 분류 시키지 않고 Maximum distance 범위 내에 들어와야만 분류되도록 Classify policy를 수정하는 것이다.

이를 위해 검증 과정을 거쳤고, D>M/1.3 정도의 Maximum distance를 가지게 되면 Test data set의 경계선에 해당하는 값들만 분류하고 중앙에 있는 값을 Fitting하지는 않게 된다.

하지만 위 값은 생성되는 Sample에 따라 다르다. Sample의 분포에 따라 Clustering 되는 군집의 Mean Vector나, 분포가 달라지기 때문에 잘 조정하는 것이 필요하다.

감사합니다.

컴퓨터 소프트웨어 학부 2016025469 서건식