알고리즘 응용(00) Lab03

201802161 조은빈

1. Goal

① Scikit-lean에서 제공하는 DBSCAN, Agglomerative Clustering 활용하여 그래프 그리기
② K-Means 클러스터링 구현 및 활용하여 그래프 그리기

2. 코드 분석

```
1  import numpy as np
2  import random
3  import copy
4  from math import *
5  import matplotlib.pyplot as plt
6  from sklearn.preprocessing import MinMaxScaler
7  from sklearn.cluster import DBSCAN
8  from sklearn.cluster import AgglomerativeClustering
9  from collections import OrderedDict
```

코드 구현에 필요한 라이브러리를 import 한다.

```
11 def read_data():
12     f = open("C:/Users/ChoEunBin/Desktop/covid-19.txt",'r',encoding = "utf-8")     # 파일 열기
13     lines = f.readlines()  # 데이터 세로 요소 길이 구하기 위해 모든 줄 읽는다
14     dataArr = np.zeros((len(lines) - 1, 2), dtype= float)  # 데이터 저장할 배열
15     f.seek(0)  # 파일 읽는 위치를 첫 줄로 이동
16     f.readline()   # 첫 줄은 버린다
17
18     index = 0
19     while True:
20         line = f.readline()
21         if line == "":
22             break
23
24         tempList = line.split('₩t')
25         dataArr[index] = tempList[5:7]
26         index += 1
27
28     f.close()
29     return dataArr
```

read_data는 파일을 읽어 필요한 데이터를 배열로 만들어 반환한다. “covid-19.txt” 파일에서 한 줄을 읽어 ‘\t’(탭)를 기준으로 리스트를 만들고 사망률, 완치율 데이터만 뽑아내서 저장한다.

```
31 def normalization(dataArr):     # 정규화
32     scaler = MinMaxScaler()
33     return scaler.fit_transform(dataArr)
```

nomalization은 데이터 값들을 MinMax로 정규화하기 위한 메소드이다.

```
35  def dbscan_func(dataArr):        # dbscan 밀도 기반 클러스터링
36      dbscan = DBSCAN(eps = 0.1,min_samples = 2)
37      return dbscan.fit_predict(dataArr)
38
39  def agglomerative_func(dataArr):    # 계층적 클러스터링
40      agg = AgglomerativeClustering(n_clusters = 8, affinity = 'Euclidean', linkage = 'complete')
41      return agg.fit_predict(dataArr)
```

dbscan_func는 Scikit-lean의 DBSCAN을 활용해 클러스터링 하는 메소드이다. DBSCAN의 파라미터로는 eps와 min_sample이 사용되었다. 어느 점을 기준으로 반경 eps 내에 점이 min_sample개 있으면 하나의 클러스터로 인식한다는 의미이다.

agglomerative_func는 Scikit-lean의 AgglomerativeClustering을 활용하여 클러스터링 하는 메소드이다. AgglomerativeClustering의 파라미터로는 n_clusters, affinity, linkage가 사용되었다. affinity = 'Euclidean' 방식으로 거리를 계산해서 n_clusters = 8개의 클러스터가 남을 때까지 클러스터를 linkage = 'complete'한다. 즉 클러스터 점 사이의 최대 거리가 가장 짧은 두 클러스터를 합친다.

```
43  def draw_graph(data, labels):   # 그래프 시각화
44      plt.figure()
45      plt.scatter(data[:,0], data[:,1], c = labels, cmap = 'rainbow')
46      plt.show()
```

draw_graph는 그래프를 시각화하기 위한 메소드이다.

```
48  def euclidean_cal(list1, list2):    # 유클리디안 거리 계산
49      return sqrt(sum(pow(a - b,2) for a, b in zip(list1, list2)))
```

euclidean_cal은 두 데이터 사이의 유클리디안 거리를 구하기 위한 메소드이다.

```
52  class KMeans:
53      def __init__(self, data, n):    # 객체 생성
54          self.data = data
55          self.n = n
56          self.cluster = OrderedDict()
57
58      def init_center(self):  # 초기 센터값 결정
59          index = random.randint(0, self.n)
60          index_list = []
61          for i in range(self.n):
62              while index in index_list:
63                  index = random.randint(0, self.n)
64              index_list.append(index)
65              self.cluster[i] = {'center': self.data[index], 'data':[]}
```

K-Means 클러스터링을 구현하기 위해 class를 만들어준다. __init__은 K-Means 객체를 생성하는 메소드이다. init_center는 초기 센터값을 랜덤하게 설정하는 메소드이다. random 함수를 통해 index를 뽑아 그에 해당하는 데이터를 center 값으로 저장해준다. 이때 만들어지는 클러스터는 center 값만 저장되어 있을 뿐 아무런 data도 매핑되지 않은 상태이다.

```
67      def clustering(self, cluster):  # 가까운 센터값 그룹에 매핑
68          for k in range(len(self.cluster)):  # 새로 클러스터링하기 위해 매핑되어 있던 데이터 비우기
69              self.cluster[k]['data'] = []
70
71          for i in range(len(self.data)):
72              min_distance = 100    # 센터값, 데이터 사이의 최소 거리
73              min_index = 100     # 최소거리인 데이터 인덱스
74              min_key = 100       # 최소거리인 센터 인덱스
75              for key, value in cluster.items():
76                  temp = euclidean_cal(value['center'], self.data[i])
77                  if min_distance > temp:
78                      min_distance = temp
79                      min_index = i
80                      min_key = key
81
82              self.cluster[min_key]['data'].append(self.data[min_index])  # 센터값 그룹에 매핑
83          return self.cluster
```

clustering은 데이터를 가까운 center 값의 그룹에 매핑하는 메소드이다. center 값을 갱신한 다음 새로 클러스터링을 할 경우를 위해 데이터가 매핑 되어있는 부분을 비우고 시작한다.

모든 데이터를 가장 가까운 center 값 그룹에 매핑하기 위해서는 우선 위에서 정의했던 euclidean_cal 메소드로 데이터와 모든 center 사이의 거리를 구한다. 거리들 중 가장 가까운 값을 찾은 다음 append 함수로 center 그룹에 데이터를 추가해준다.

```
85      def update_center(self):    # 센터값 갱신
86          for i in range(len(self.cluster)):
87              self.cluster[i]['center'] = (sum(self.cluster[i]['data']))/len(self.cluster[i]['data'])
```

update_center는 center 값을 갱신하기 위한 메소드이다. center 값은 각 그룹의 평균값을 계산해 갱신해준다.

```
89      def update(self):   # 센터값 갱신 및 클러스터링
90          while True:
91              state = False
92              current_cluster = copy.deepcopy(self.cluster)
93              self.update_center()
94              self.clustering(self.cluster)
95              # 기존 센터값과 클러스터링 후 센터값이 모두 동일할 경우 업데이트 종료
96              for i in range(len(self.cluster)):
97                  if np.array_equal(current_cluster[i]['center'], self.cluster[i]['center']):
98                      state = True
99                  else:
100                     state = False
101                     break
102             if(state):
103                 break
```

update는 데이터가 더 이상 새로 매핑되지 않을 때까지 center 값을 갱신하고 클러스터링 하는 메소드이다. 기존의 클러스터를 current_cluster에 복사해두고 클러스터링 후 값과 이를 비교한다. 복사할 때 새로 업데이트 되는 값이 current_cluster에 영향을 미치지 못하도록 deepcopy 함수를 통해 복사한다. 기존 클러스터의 center 값과 새로운 center 값이 일치한다면, 더 이상 데이터가 새로 매핑되지 않고 클러스터링이 완료되었다는 의미이므로 업데이트를 종료한다.

```
107     def fit(self):  # 외부에서 실행 호출하는 함수
108         self.init_center()
109         self.cluster = self.clustering(self.cluster)
110         self.update()
111 
112         result, labels = self.get_result(self.cluster)
113         draw_graph(result, labels)
```

fit은 구현한 KMeans 클래스로 K-Means 클러스터링을 하기 위해 필요한 메소드를 모아둔 메소드이다. 외부에서 fit만 호출하면 K-Means 클러스터링을 할 수 있다.

```
115     def get_result(self, cluster):  # 그래프 그리기 위해 데이터 가공
116         result = []
117         labels = []
118         for key, value in cluster.items():
119             for item in value['data']:
120                 labels.append(key)
121                 result.append(item)
122         return np.array(result), labels
```

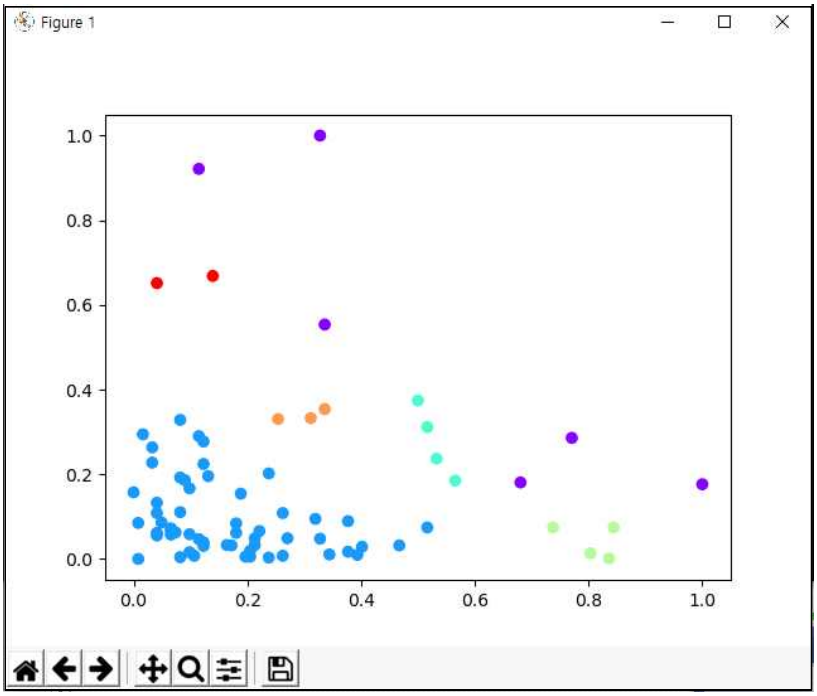
get_result는 그래프를 그리기 위해 데이터를 가공하는 메소드이다.

```
124 def main():
125     Arr = read_data()
126     nomal_Arr = normalization(Arr)
127     draw_graph(nomal_Arr,dbscan_func(nomal_Arr))
128     draw_graph(nomal_Arr,agglomerative_func(nomal_Arr))
129     KMeans_test = KMeans(nomal_Arr,8)
130     KMeans_test.fit()
131 
132 if __name__ == '__main__':
133     main()
```

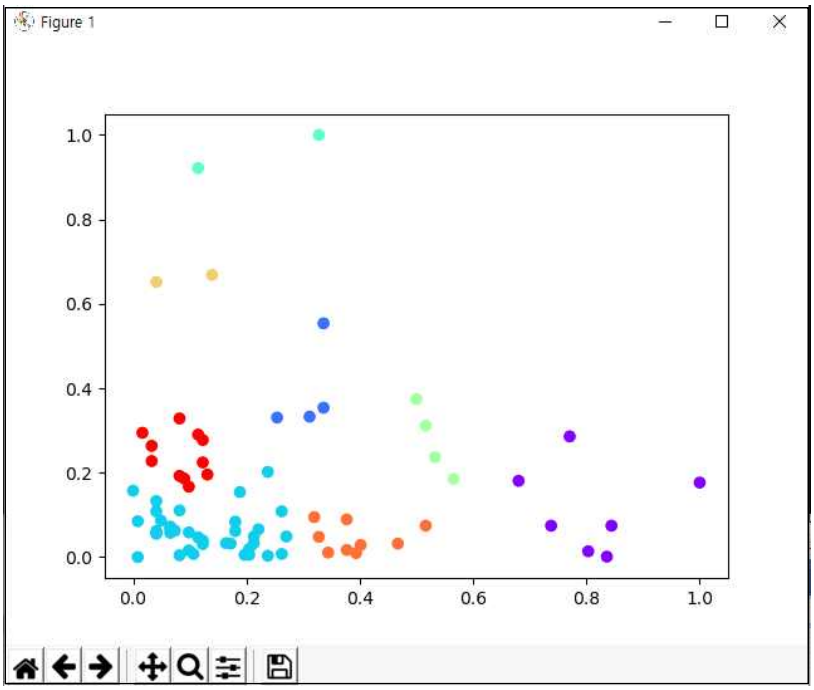
main은 클러스터링들을 실행하기 위한 코드를 작성한 메소드이다.

3. 실행 결과

- DBSCAN : 보라색 점들이 노이즈에 해당하는 점들이다.



- 계층적 클러스터링



- K-Means