알고리즘 응용(00) Lab02

201802161 조은빈

1. Goal

1. 코사인 거리 함수 활용하여 그래프 그리기
2. 맨하탄, 유클리디안 구현 및 활용하여 그래프 그리기
3. 정규화 후 1, 2 반복하기

2. 코드 분석

```
(base) C:₩Users₩ChoEunBin>activate homework

(homework) C:₩Users₩ChoEunBin>conda install scikit-learn
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

가상환경에 scikit-learn 라이브러리를 설치한다.

```
1    import numpy as np
2    import matplotlib.pyplot as plt
3    from math import *
4    from sklearn.metrics.pairwise import cosine_distances
5    from sklearn.preprocessing import MinMaxScaler
```

필요한 라이브러리들을 import한다.

```
7    def read_data():
8        f = open("C:/Users/ChoEunBin/Desktop/seoul_tax.txt",'r',encoding = "utf-8")     # 파일 열기
9        line = f.readline()     # 데이터 가로 요소 길이 구하기 위해 한 줄 읽는다
10       lines = f.readlines()   # 데이터 세로 요소 길이 구하기 위해 첫 줄 이후 모든 줄 읽는다
11       dataArr = np.zeros((len(lines), len(line.split())-1), dtype= int)   # 데이터 저장할 배열
12       f.seek(0)   # 파일 읽는 위치를 첫 줄로 이동
13       f.readline()    # 첫 줄은 버린다
14
15       index = 0
16       while True:
17           line = f.readline()
18           if line == "":
19               break
20
21           tempList = line.split()
22           tempList = tempList[1:]
23           tempArr = np.array(tempList, dtype= int)
24           dataArr[index] = tempArr + dataArr[index]
25           index += 1
26
27       f.close()
28       return dataArr
```

파일 오픈 후 숫자 데이터들만 모아 배열에 저장한다. 배열을 리턴한다.

```
def cosine_main(dataArr):   # 모든 구의 코사인 거리 구하기
    cosine_Arr = np.zeros((len(dataArr), len(dataArr)), dtype= float)
    for i in range(len(cosine_Arr)):
        for k in range(len(cosine_Arr)):
            cosine_Arr[i][k] = cosine_distances(dataArr[[i]], dataArr[[k]])
    show_graph(cosine_Arr, 'cosine distance')
```

cosine_main 함수는 모든 구의 코사인 거리를 구하는 함수이다. cosine_Arr는 각 구 간의 코사인 거리를 저장할 배열이다. 이중for문을 통해 배열을 채운다. 코사인 거리를 구하는 함수로는 sklearn 라이브러리의 cosine_distances 함수를 사용한다. cosine_distances()는 파라미터로 이차원 배열을 받기 때문에 dataArr[[i]] 와 같은 형식으로 넣어주었다.

```
def manhattan_cal(list1, list2): # 맨하탄 거리 계산
    return (sum(abs(b-a) for a, b in zip(list1, list2)))

def manhattan_main(dataArr):    # 모든 구의 맨하탄 거리 구하기
    manhattan_Arr = np.zeros((len(dataArr), len(dataArr)), dtype= int)
    for i in range(len(manhattan_Arr)):
        for k in range(len(manhattan_Arr)):
            manhattan_Arr[i][k] = manhattan_cal(dataArr[i],dataArr[k])
    show_graph(manhattan_Arr, 'manhattan distance')
```

manhattan_main 함수는 모든 구의 맨하탄 거리를 구하는 함수이다. manhattan_Arr는 각 구 간의 맨하탄 거리를 저장할 배열이다. 이중for문을 통해 배열을 채운다. manhattan_cal 함수는 맨하탄 거리를 계산하는 함수이다. 맨하탄 거리 공식은 $d(p,q) = \sum_{i=1}^{n} |p_i - q_i|$ 이다. 즉 서로 대응되는 요소의 차를 모두 더한 값이다. 행 두 개를 파라미터로 받아 두 행의 서로 대응되는 데이터 요소의 차를 모두 더한 값을 리턴한다.

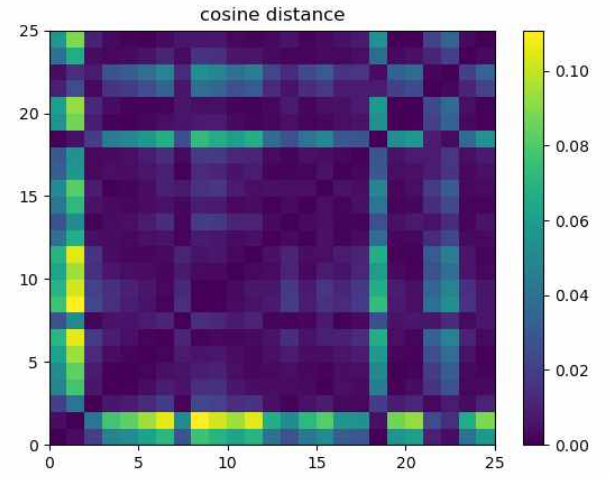
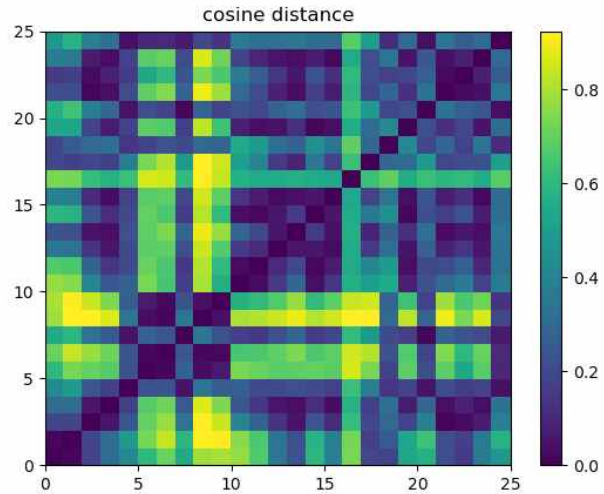
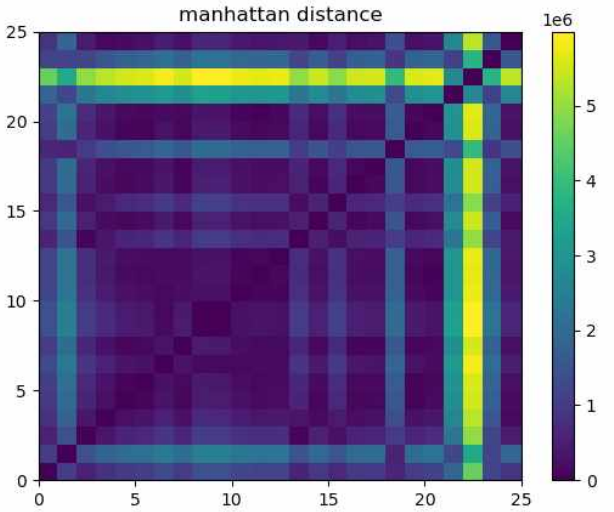
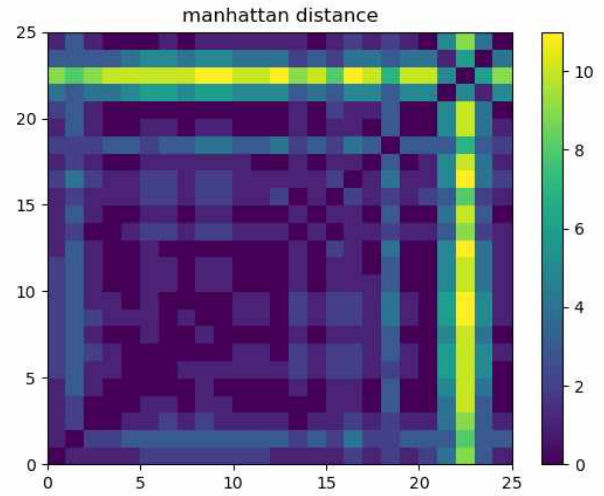
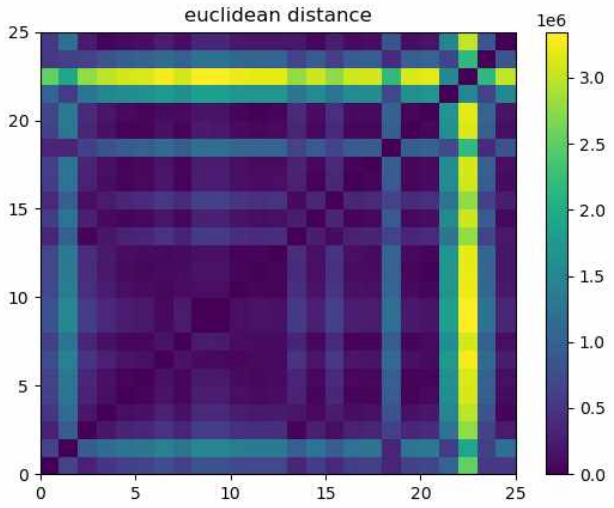
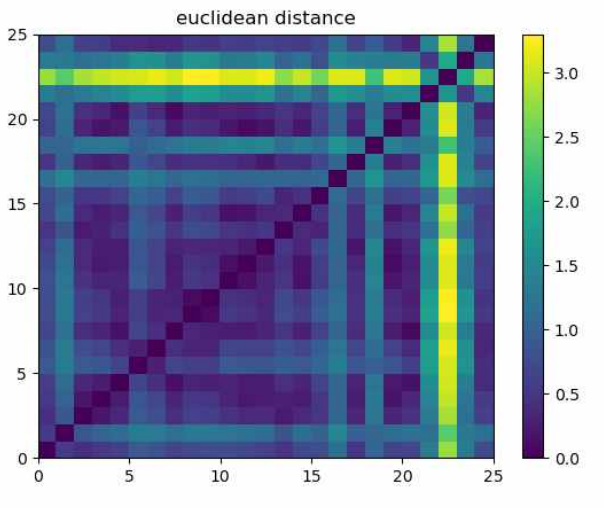
```
def euclidean_cal(list1, list2):    # 유클리디안 거리 계산
    return sqrt(sum(pow(a-b,2) for a, b in zip(list1, list2)))

def euclidean_main(dataArr):    # 모든 구의 유클리디안 거리 구하기
    euclidean_Arr = np.zeros((len(dataArr), len(dataArr)), dtype= float)
    for i in range(len(euclidean_Arr)):
        for k in range(len(euclidean_Arr)):
            euclidean_Arr[i][k] = euclidean_cal(dataArr[i],dataArr[k])
    show_graph(euclidean_Arr, 'euclidean distance')
```

euclidean_main 함수는 모든 구의 유클리디안 거리를 구하는 함수이다. euclidean_Arr는 각 구 간의 유클리디안 거리를 저장할 배열이다. 이중for문을 통해 배열을 채운다. euclidean_cal 함수는 유클리디안 거리를 계산하는 함수이다. 유클리디안 거리 공식은 $d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$ 이다. 이를 코드로 구현하면 위와 같다.

```
59  def normalization(dataArr):      # 정규화
60      scaler = MinMaxScaler()
61      return scaler.fit_transform(dataArr)
62
63  def show_graph(result_data, graph_title):    # 그래프 출력 함수
64      plt.pcolor(result_data)
65      plt.title(graph_title)
66      plt.colorbar()
67      plt.show()
68
69
70  def main():
71      dataArr = read_data()
72      print("1. 코사인 거리 그래프 (아무 키나 누르세요)")
73      input()
74      cosine_main(dataArr)
75      print("2. 맨하탄 거리 그래프 (아무 키나 누르세요)")
76      input()
77      manhattan_main(dataArr)
78      print("3. 유클리디안 거리 그래프 (아무 키나 누르세요)")
79      input()
80      euclidean_main(dataArr)
81      normal_dataArr = normalization(dataArr)
82      print("4. 정규화 코사인 거리 그래프 (아무 키나 누르세요)")
83      input()
84      cosine_main(normal_dataArr)
85      print("5. 정규화 맨하탄 거리 그래프 (아무 키나 누르세요)")
86      input()
87      manhattan_main(normal_dataArr)
88      print("6. 정규화 유클리디안 거리 그래프 (아무 키나 누르세요)")
89      input()
90      euclidean_main(normal_dataArr)
91
92  if __name__ == '__main__':
93      main()
```

MinMaxScaler() 함수를 통해 데이터들을 정규화한 다음 거리를 다시 구한다.

3. 실행 결과

정규화 전	정규화 후
	
	
	

맨하탄과 유클리디안 거리 그래프는 정규화 전과 후 모양에 큰 차이가 나타나지 않는다. 하지만 코사인 거리 그래프는 크게 차이가 나타났다.