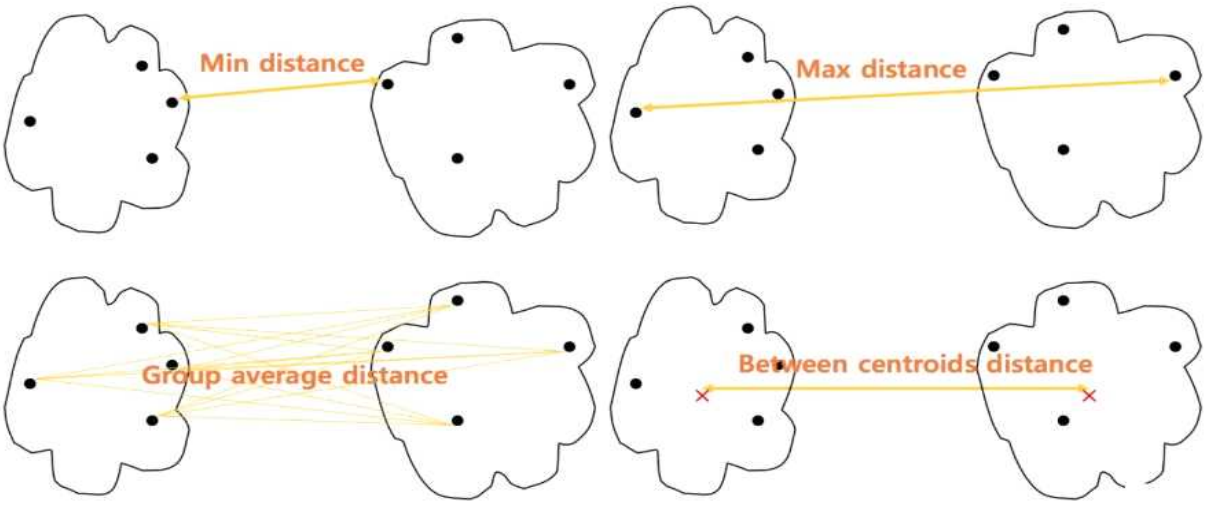
정보검색과
데이터마이닝

hw5 - 문서 클러스터링

20142772 최승호

1. 내가 구현한 SLINK 알고리즘 설명



slink는 첫 번째 그림으로, 클러스터간의 각각의 요소들끼리 유사도가 제일 높은 것(distance가 제일 낮은 것)이 있는 클러스터들 끼리 묶어주는 것을 뜻한다.
전 과제에서 했던 623개 문서의 similarity_upper_triangle을 이용하여 slink 알고리즘을 구성하겠다.

내가 짠 slink알고리즘 설명
1. upper_triangle을 읽어서 유사도값이 들어있는 리스트를 만들어 역순으로 정렬해준다
-> 유사도값이 높은 것부터 반복문을 돌린다.

2. 해당 유사도 값이 있는 document vector를 찾는다.

3. document vector에서 유사도 값이 어떤 문서 A(기준 문서, row에 해당됨) + 문서 B(클러스터링 될 문서, col에 해당됨)의 관계인지 row, col을 찾는다.

4. 문서 A, B가 같은 클러스터인지 확인한다.

5. 같은 클러스터면 다음 문서벡터를 확인하고, 다른 클러스터면 클러스터링을 해준다.

6. 클러스터링은 2차원 배열로 선언하고 A가 있는 배열에 문서 값을 append해주고 B가 있는 배열을 삭제한다.

7. 위 알고리즘을 정해진 클러스터링 개수가 될 때 까지 반복한다.

자세한 내용은 코드에서 설명하겠다. 대략의 루틴은 이렇게 진행된다.

2. 코드 설명

csv파일 읽기

```
import csv

f = open('similarity_matrix.csv', 'r', encoding='utf-8')
rdr = csv.reader(f)

doc_similarity_matrix = list()
for line in rdr:
    # string 전부 float로 변환
    line = list(map(float, line))
    doc_similarity_matrix.append(line)
f.close()

print(doc_similarity_matrix[2])
```

```
[0.005876797, 0.060260286, 1.0, 0.005293993, 0.003464489, 0.003897879, 0.008538554, 0.003962145, 0.02294459, 0.004631465, 0.010294364, 0.00
6284358, 0.00553159, 0.013381239, 0.001238908, 0.008709711, 0.00843252, 0.006133089, 0.008140105, 0.004970235, 0.015583814, 0.011098661, 0.
005601112, 0.002662488, 0.006872075, 0.002535221, 0.00438514, 0.012429437, 0.003486727, 0.04801624, 0.003975084, 0.006258383, 0.003070668,
0.005768542, 0.005966112, 0.003471749, 0.00501224, 0.015789258, 0.002127434, 0.003020226, 0.001954352, 0.008335771, 0.017737091, 0.0049678
2, 0.008189048, 0.010670966, 0.002554352, 0.0060382, 0.012039503, 0.01112037, 0.006492111, 0.001211848, 0.008467774, 0.008050135, 0.0045887
38, 0.003565357, 0.014353612, 0.010939392, 0.006092214, 0.011368688, 0.005436538, 0.007603526, 0.010085339, 0.004937398, 0.002697209, 0.010
5962, 0.006246206, 0.008820089, 0.003325016, 0.002649781, 0.007179264, 0.006245563, 0.024271019, 0.0132371, 0.016014363, 0.017151018, 0.016
393895, 0.010384169, 0.007939241, 0.01281647, 0.01437209, 0.00449846, 0.002311996, 0.027692951, 0.033805073, 0.083806231, 0.009428536, 0.00
9820236, 0.011091475, 0.017507643, 0.002632976, 0.00760043, 0.008388283, 0.006770558, 0.011646446, 0.002016904, 0.006608298, 0.002598137,
```

전 과제에서 했던 623개 문서출력 파일을 읽어와 similarity_matrix를 재 생성한다.
2번째 인덱스의 document vector를 출력하면 623개의 유사도가 나온다.

slink하기 위한 자료구조 설정

```
index = 0
# 클러스터링 하기위한 리스트 선언
# [0], [1], [2] ... 을 가진 2차원 배열 선언
clustering = list()
for i in range(623):
    clustering.append([i])

print(clustering[:3])

# upper triangle을 전부다 리스트로 받은다음에 sort를 해주자
# 유사도를 역순의 리스트로 저장하기
inverse_sorted_similarity = []
index = 1
for doc_vector in doc_similarity_matrix:
    inverse_sorted_similarity = inverse_sorted_similarity + doc_vector[index:]
    index += 1

inverse_sorted_similarity.sort(reverse = True)
print(inverse_sorted_similarity[:10])
```

```
[[0], [1], [2]]
[0.998958879, 0.734137663, 0.693477377, 0.668761286, 0.666535717, 0.636236495, 0.622331558, 0.609812854, 0.599560559, 0.59173583]
```

일단 클러스터링을 위한 [[0], [1], [2] ... [622]]의 형태인 2차원 배열을 선언한다.
방금 만들었던 similarity_matrix를 활용하여 upper triangle만 활용해서 유사도 리스트를 만들어 모든 값을 추가해준다.
그리고 그 값을 역순으로 정렬해준다.
상위 10개 값을 출력해보면 0.99 0.73 ... 이렇게 나온다.

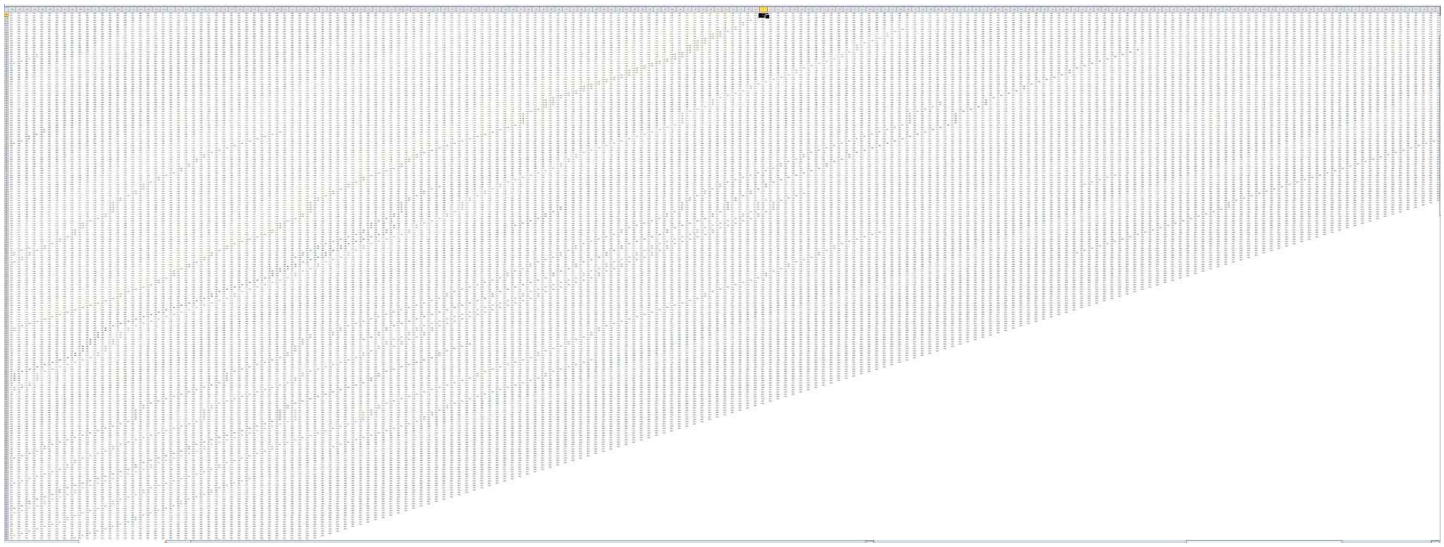
Slink Clustering

클러스터끼리 유사도가 가장높은 document가 있는 것끼리 묶어주기

```
f = open('clustering.csv', 'w', newline='')
csv_writer = csv.writer(f)


# 전체에서 제일 높은 유사도 값을 찾아 클러스터링 해준다
# 그러면 각 doc_vector를 돌 때마다 들어오는 해당 유사도 값이 있는지 확인해주자

# slink 알고리즘 start
n = 0
for similarity in inverse_sorted_similarity:

    #클러스터링 개수 정해주기
    if(len(clustering) == 5):
        break

    row = 0
    # loop를 중단해주기 위한 flag이다.
    merge = False

    for doc_vector in doc_similarity_matrix:

        # 유사도 값이 문서벡터에 있으면
        if similarity in doc_vector:

            # row = 기준이 되는 문서번호, col = 클러스터링 될 문서번호
            col = doc_vector.index(similarity)

            # row col을 받아서 clustering 해주기
            for row_cluster in clustering:

                # row 번호가 들어있는 클러스터(기준이 되는 클러스터)를 찾자
                if row in row_cluster:

                    # 같은 클러스터면 다음 문서벡터 확인
                    # 유사도가 같은 문서가 있을 수도 있으므로 다음 유사도를 넘어가면 잘못된 클러스터링이 이루어 질수도 있다.
                    if col in row_cluster:
                        merge = True
                        break

                    # 다른 클러스터면 합치고 뒤에있는 클러스터(col) 삭제
                    else:
                        for col_cluster in clustering:
                            if col in col_cluster:

                                #합치고 지우기
                                row_cluster += col_cluster
                                clustering.remove(col_cluster)

                                # csv 파일 출력
                                csv_writer.writerow(clustering)

                                # loop를 멈추기 위한 break문
                                # 여기서 다음 유사도로 넘어가야한다
                                merge = True
                                n += 1
                                print(n, "번째 클러스터링 진행중")
                                break

                if(merge == True):
                    break
        if merge == True:
            break

        row += 1
```

위에서 설명한 알고리즘을 토대로 코드를 짜보았다.
csv파일에 출력해가며 클러스터링이 되는 것을 확인할 수 있다.
가장 큰 loop로 역순으로 정렬된 유사도 리스트를 돌리고 그다음 해당 유사도를 찾기위해 similarity_matrix를 loop돌려 doc_vector에 해당 유사도가 있는지 계속 체크한다.
나머지 코드 세세한 설명은 주석으로 달아 놓았다.

3. 출력 결과



클러스터가 1개씩 줄어드는 것을 확인할 수 있다.

A617 [160, 178, 19, 38, 330, 111, 242, 268, 495, 324, 497, 395, 115, 194, 250, 576, 28, 203, 309, 326, 39, 167, 293, 401, 126, 448, 449, 129, 331, 285,

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
[19, 38, 3	[111]	[115, 194,	[160]	[169]	[178]	[242]	[268]	[280]	[317]	[324]	[363]	[366]	[395]	[495]	[497]	[581]	[585]
[19, 38, 3	[111]	[115, 194,	[160]	[169]	[178]	[242]	[268]	[280]	[317]	[324]	[366]	[395]	[495]	[497]	[581]	[585]	[602]
[19, 38, 3	[111]	[115, 194,	[160]	[169]	[178]	[242]	[268]	[280]	[317]	[324]	[366]	[395]	[495]	[497]	[581]	[585]	[602]
[19, 38, 3	[111]	[160]	[169]	[178]	[242]	[268]	[280]	[317]	[324]	[366]	[395, 115,	[495]	[497]	[581]	[585]	[602]	[612]
[19, 38, 3	[111]	[160]	[169]	[178]	[242]	[268]	[280]	[317]	[324]	[366]	[395, 115,	[495]	[497]	[585]	[602]	[612]	[621]
[19, 38, 3	[111]	[160]	[169]	[178]	[242]	[268]	[280]	[317]	[324]	[395, 115,	[495]	[497]	[585]	[602]	[612]	[621]	
[19, 38, 3	[111]	[160]	[169]	[178]	[242]	[268]	[280]	[317]	[324]	[495]	[497, 395,	[585]	[602]	[612]	[621]		
[19, 38, 3	[111]	[160]	[169]	[178]	[242]	[268]	[280]	[317]	[324, 497,	[495]	[585]	[602]	[612]	[621]			
[19, 38, 3	[111]	[160]	[169]	[178]	[242]	[268]	[280]	[317]	[495, 324,	[585]	[602]	[612]	[621]				
[19, 38, 3	[111]	[160]	[169]	[178]	[242]	[268, 495,	[280]	[317]	[585]	[602]	[612]	[621]					
[19, 38, 3	[111]	[160]	[169]	[178]	[242, 268,	[280]	[317]	[585]	[602]	[612]	[621]						
[19, 38, 3	[111]	[160]	[169]	[178]	[242, 268,	[280]	[317]	[602]	[612]	[621]							
[19, 38, 3	[111, 242,	[160]	[169]	[178]	[280]	[317]	[602]	[612]	[621]								
[19, 38, 3	[160]	[169]	[178]	[280]	[317]	[602]	[612]	[621]									
[160]	[169]	[178, 19, 3	[280]	[317]	[602]	[612]	[621]										
[160, 178,	[169]	[280]	[317]	[602]	[612]	[621]											
[160, 178,	[169]	[280]	[317]	[602]	[612]												
[160, 178,	[169]	[280]	[317]	[602]													

문서가 여러개인 클러스터가 여러개인 클러스터랑 묶이는 것도 확인 할 수 있고,
문서가 1개인 클러스터가 여러개인 클러스터랑 묶이는 것도 확인 할 수 있다.
클러스터가 5개 되면 종료된다.

4. Scikit learan 라이브러리 사용

output.csv파일을 통해 데이터 읽어오기 + 필요한 라이브러리 import ¶

```
from sklearn import datasets
import pandas as pd


similarity_matrix = pd.read_csv('similarity_matrix_lib.csv')

similarity_matrix.head()
```

	0	1	2	3	4	5	6	7	8	9	...	613	614	615	616	617
0	1.000000	0.005433	0.005877	0.017805	0.096693	0.013086	0.015359	0.005795	0.052837	0.036604	...	0.003555	0.007274	0.008734	0.004565	0.008222
1	0.005433	1.000000	0.060260	0.004626	0.002105	0.006940	0.006062	0.007252	0.021649	0.006153	...	0.032232	0.039814	0.011712	0.018891	0.312353
2	0.005877	0.060260	1.000000	0.005294	0.003464	0.003898	0.008539	0.003962	0.022945	0.004631	...	0.049477	0.078306	0.013132	0.011280	0.110643
3	0.017805	0.004626	0.005294	1.000000	0.028014	0.023914	0.005678	0.016349	0.029457	0.042855	...	0.004437	0.016790	0.025346	0.019621	0.010065
4	0.096693	0.002105	0.003464	0.028014	1.000000	0.005182	0.004470	0.022644	0.091417	0.079304	...	0.005525	0.004637	0.023728	0.002415	0.007219

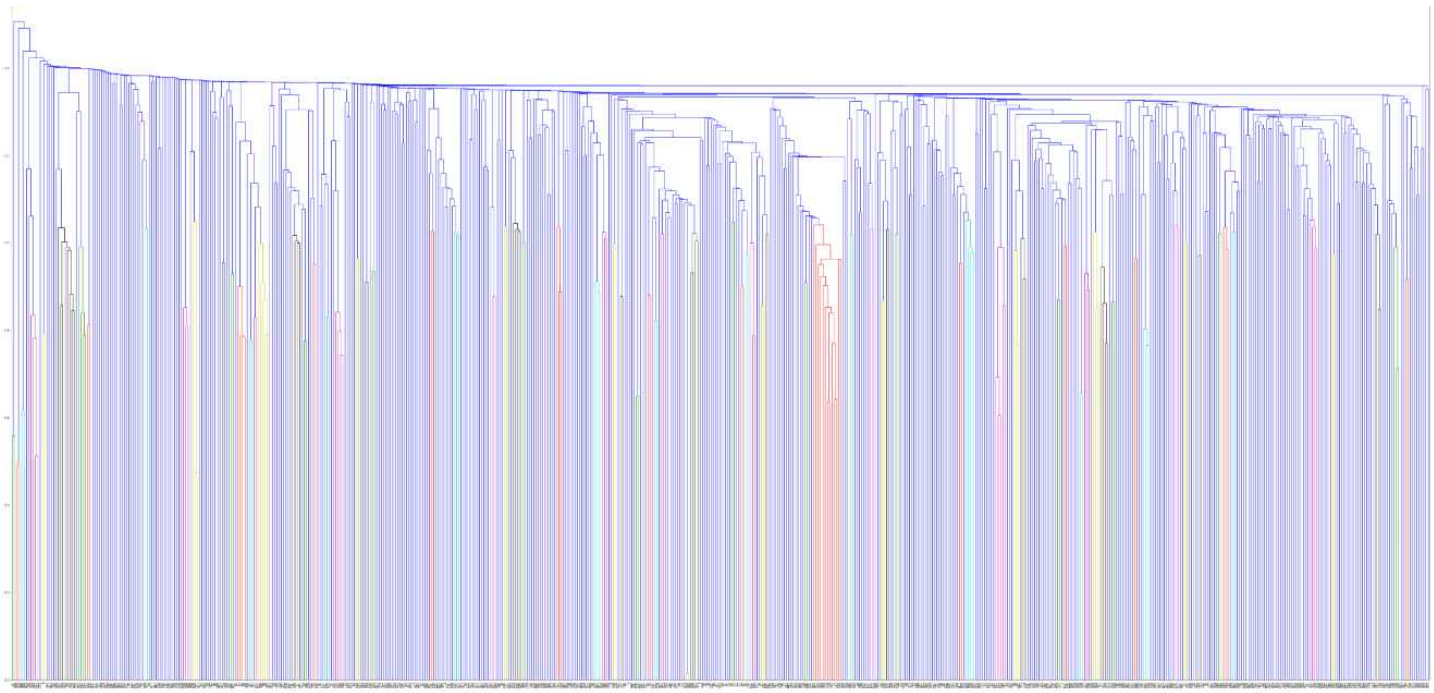
5 rows × 623 columns

pandas 라이브러리를 통해서 csv파일을 pd.array로 읽어온다.

계층적 클러스터링

```
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt

mergings = linkage(similarity_matrix, method = 'single')

# Plot the dendrogram, using varieties as labels
plt.figure(figsize=(80,40))
dendrogram(mergings,
#            labels = labels.as_matrix(),
           leaf_rotation= 90,
           leaf_font_size=10,
)
plt.show()
```

hierarchy라이브러리 사용하여 linkage함수를 이용한다! method에 따라 single, complete, k-means 전부 가능하다!



dendogram함수와 matplot라이브러리를 통하여 한번 출력해 보았다 623개의 문서라 알아보긴 쉽지 않지만 대략 클러스터가 어떻게 되가고 있는지 확인할 수 있다.

최종적으로 분리된 클러스터링 정보

```
from sklearn.cluster import AgglomerativeClustering

ac  = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = "single")
result = ac.fit_predict(similarity_matrix)
print(result)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 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 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 0 0 0 0 0 0
 1 1 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 3 0 0 2 0 0 0 0 0 0 1 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 1 0 0 0 0 1 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 4 4 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 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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]
```

다른 함수인데 affinity를 euclidean, cosine 등등으로 설정해서 linkage되는 값의 기준을 바꿀수 도 있다. 클러스터 개수를 5개 기준으로 하면 0 1 2 3 4 로 623개가 클러스터 된것들로 나눈 것을 확인할 수 있다.

5. 후기

라이브러리를 사용하여 클러스터를 한 것과 내 것의 결과가 약간의 차이는 있지만 대략 클러스터링 하는 문서들이 비슷하다. 모든 문서를 확인하지 않고 유사도 값을 먼저 정렬하여 반복문을 돌려 시간적인 복잡도를 줄여 만들어서 뿌듯한 알고리즘이였다.
그다음 scikit learn 라이브러리를 사용하여 결과 비교도 해보고 나중에 클러스터링을 할려면 라이브러리 사용법을 익숙하게 하는 것이 중요한 것 같다.
그리고 scikit learn 라이브러리들이 numpy, matplot 다양한 라이브러리를 쓰는데 데이터 분석과 머신러닝에서 필수적인 라이브러리라 사용법을 익혀야겠다고 생각했다.