

# Data Analytics Week 9 Assignment

202011431 산업공학과 차승현

## Latent Semantic Analysis



# Analysis Procedure

1) 필요한 모듈 pandas, numpy, csv를 import하고, 제시된 csv파일을 불러온다.

```
In [5]: import pandas as pd
import numpy as np
import csv
```

```
In [3]: df = pd.read_csv(r'data_week9.csv')
df
```

Out [3]:

	Unnamed: 0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
0	database	11	21	12	0	37	2	0	3	1	6
1	SQL	21	10	0	7	31	0	0	21	5	0
2	index	9	0	5	2	20	0	1	0	11	0
3	regression	3	5	2	2	0	18	32	11	21	8
4	likelihood	0	3	0	0	3	7	12	4	27	4
5	linear	3	0	0	4	0	16	21	2	16	15

**constraints 1) Term-document matrix를 TF-IDF matrix로 변환한 후 LSA를 수행할 것**

2) row와 column의 name을 제외한 정수 value의 값들을 행렬의 형태로 변형해준다.

```

In [6]: f = open('data_week9.csv', encoding='utf-8')
        reader = csv.reader(f)
        tf_lst = []
        for line in reader:
            print(line)
            tf_lst.append(line[1:])
        f.close()

['', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9', 'D10']
['database', '11', '21', '12', '0', '37', '2', '0', '3', '1', '6']
['SQL', '21', '10', '0', '7', '31', '0', '0', '21', '5', '0']
['index', '9', '0', '5', '2', '20', '0', '1', '0', '11', '0']
['regression', '3', '5', '2', '2', '0', '18', '32', '11', '21', '8']
['likelihood', '0', '3', '0', '0', '3', '7', '12', '4', '27', '4']
['linear', '3', '0', '0', '4', '0', '16', '21', '2', '16', '15']

In [10]: tf_lst_1 = tf_lst[1:]

        A = np.array(tf_lst_1).astype(np.float64)
        A

Out[10]: array([[11., 21., 12., 0., 37., 2., 0., 3., 1., 6.],
                [21., 10., 0., 7., 31., 0., 0., 21., 5., 0.],
                [ 9., 0., 5., 2., 20., 0., 1., 0., 11., 0.],
                [ 3., 5., 2., 2., 0., 18., 32., 11., 21., 8.],
                [ 0., 3., 0., 0., 3., 7., 12., 4., 27., 4.],
                [ 3., 0., 0., 4., 0., 16., 21., 2., 16., 15.]])

```

3) 각 term이 몇 개의 documnet에서 빈도를 나타내는지 count라는 리스트에 저장하고, idf matrix를 구한다.

```

In [36]: count = []

         for i in range(len(tf_lst_1)):
             counts = 0
             for j in range(10):
                 if A[i][j] > 0:
                     counts += 1
             count.append(counts)

In [37]: count

Out [37]: [8, 6, 6, 9, 7, 7]

In [39]: idf = list(map(lambda i : np.log(1+(10/i)), count))
         idf

Out [39]: [0.8109302162163288,
           0.9808292530117263,
           0.9808292530117263,
           0.7472144018302211,
           0.8873031950009029,
           0.8873031950009029]

```

4) tf와 idf를 곱하여 TF-IDF matrix를 구한다.

```

In [57]: B = A.copy()

In [58]: for i in range(len(idf)):
         B[i] = B[i] * idf[i]

In [59]: B

Out [59]: array([[ 8.92023238, 17.02953454,  9.73116259,  0.          , 30.004418  ,
                    1.62186043,  0.          ,  2.43279065,  0.81093022,  4.8655813 ],
                  [20.59741431,  9.80829253,  0.          ,  6.86580477, 30.40570684,
                    0.          ,  0.          , 20.59741431,  4.90414627,  0.          ],
                  [ 8.82746328,  0.          ,  4.90414627,  1.96165851, 19.61658506,
                    0.          ,  0.98082925,  0.          , 10.78912178,  0.          ],
                  [ 2.24164321,  3.73607201,  1.4944288 ,  1.4944288 ,  0.          ,
                  13.44985923, 23.91086086,  8.21935842, 15.69150244,  5.97771521],
                  [ 0.          ,  2.66190959,  0.          ,  0.          ,  2.66190959,
                    6.21112237, 10.64763834,  3.54921278, 23.95718627,  3.54921278],
                  [ 2.66190959,  0.          ,  0.          ,  3.54921278,  0.          ,
                  14.19685112, 18.6333671 ,  1.77460639, 14.19685112, 13.30954793]])

```

5) TF-IDF Matrix에서 numpy.linalg.svd 모듈을 사용하여 U, S,  $V^T$  행렬을

구해준다.

```
In [60]: U, s, VT = np.linalg.svd(B, full_matrices = True)
```

```
In [61]: print('U: ', U)
print('s: ', s)
print('VT: ', VT)
```

```
U: [[-0.50349475 -0.32826357  0.68444393  0.31275271 -0.25236036  0.09366201]
     [-0.63170778 -0.34257728 -0.67503801  0.06155391  0.00897494  0.15504947]
     [-0.34386524 -0.09961085  0.21847092 -0.57492684  0.52054032 -0.47180689]
     [-0.30910561  0.56140574 -0.1163579  0.31240244 -0.27710086 -0.6335282 ]
     [-0.26287821  0.40689774  0.06217033 -0.62813665 -0.47525904  0.37554937]
     [-0.25405469  0.53312226  0.10356804  0.27514552  0.60217558  0.44965547]]
s: [61.76133769 47.43465416 18.28024796 16.03116557  9.66589321  6.15247347]
VT: [[-3.54711714e-01 -2.69178994e-01 -1.14114934e-01 -1.03225726e-01
      -6.76148052e-01 -1.65371588e-01 -2.47099258e-01 -2.94050493e-01
      -3.55743924e-01 -1.39438426e-01]
     [-1.72576570e-01 -1.21634631e-01 -5.99542991e-02  3.87214945e-03
      -4.45593242e-01  3.60799078e-01  5.81691941e-01 -1.79227360e-02
      4.87093680e-01  2.17109487e-01]
     [-3.20304126e-01  2.60694574e-01  4.13449528e-01 -2.19494695e-01
      2.44114794e-01  7.66708777e-02  1.30487783e-03 -6.99709761e-01
      4.02399525e-02  2.31603132e-01]
     [ 2.59020490e-02  3.38396290e-01  4.30901566e-02  4.60490656e-02
      -1.05706027e-01  2.94038285e-01  3.33389468e-01  1.78112142e-01
      -7.41529911e-01  3.00779652e-01]
     [ 3.63191861e-01 -6.73493471e-01 -3.28020704e-02  2.90287118e-01
      1.70402316e-01  1.51133906e-01  4.65692682e-03 -3.43976647e-01
      -1.78924003e-01  3.56260864e-01]
     [-5.83431953e-02  2.84204790e-01 -3.81819207e-01  1.28107101e-01
      -1.18797556e-01  5.64515277e-02 -5.25586535e-01  5.60992153e-02
      1.92726614e-01  6.47915783e-01]
     [-2.08707834e-01  1.81539436e-02 -7.88079085e-01 -1.72513354e-01
      3.48345743e-01 -1.24675717e-01  3.12942962e-01 -2.32499863e-01
      -1.01460471e-01 -7.81631029e-02]
     [-5.91915275e-02 -3.43505060e-01  4.79448752e-02 -8.09734833e-01
      1.27064323e-01  1.93736248e-01 -1.35338499e-01  3.15787195e-01
      -4.48617211e-02  2.16337039e-01]
     [ 7.44240319e-01  2.64442554e-01 -1.07035109e-01 -3.90574890e-01
      -3.03780880e-01 -1.36947350e-01  6.80566425e-02 -3.05079070e-01
      5.68839460e-02 -1.38958856e-02]
     [-6.97335840e-02 -8.74814313e-02  1.70000698e-01 -1.93100067e-02
      -2.41235828e-02 -8.07637759e-01  3.03779191e-01  1.53549227e-01
      -4.62614245e-04  4.35305686e-01]]
```

6) 대각 행렬의 크기인  $6 \times 10$ 의 임의의 행렬을 생성한 후, 특이값을 대각행렬에 삽입한다.

```
print('대각 행렬 S :')
print(S)
```

```

대각 행렬 S :
[[61.76133769  0.          0.          0.          0.          0.
  0.          0.          0.          0.          ]
 [ 0.          47.43465416  0.          0.          0.          0.
  0.          0.          0.          0.          ]
 [ 0.          0.          18.28024796  0.          0.          0.
  0.          0.          0.          0.          ]
 [ 0.          0.          0.          16.03116557  0.          0.
  0.          0.          0.          0.          ]
 [ 0.          0.          0.          0.          9.66589321  0.
  0.          0.          0.          0.          ]
 [ 0.          0.          0.          0.          0.          6.15247347
  0.          0.          0.          0.          ]]
```

constraints 2) 2개의 Singular values를 활용

7) S벡터, U벡터, V transpose 벡터에서 상위 2개의 singular values를 추출한다.

```
In [64]: # 특이값 상위 2개만 보존  
S = S[:, :2]
```

```
print('대각 행렬 S :')  
print(S.round(2))
```

```
대각 행렬 S :  
[[61.76  0.  ]  
 [ 0.   47.43]]
```

```
In [65]: U = U[:, :2]  
print('행렬 U :')  
print(U.round(2))
```

```
행렬 U :  
[[-0.5  -0.33]  
 [-0.63 -0.34]  
 [-0.34 -0.1 ]  
 [-0.31  0.56]  
 [-0.26  0.41]  
 [-0.25  0.53]]
```

```
In [66]: VT = VT[:, :]  
print('직교행렬 VT :')  
print(VT.round(2))
```

```
직교행렬 VT :  
[[-0.35 -0.27 -0.11 -0.1  -0.68 -0.17 -0.25 -0.29 -0.36 -0.14]  
 [-0.17 -0.12 -0.06  0.   -0.45  0.36  0.58 -0.02  0.49  0.22]]
```

```
In [68]: A = B
```

```
In [69]: A_prime = np.dot(np.dot(U,S), VT)
print(A)
print(A_prime.round(2))
```

```
[[ 8.92023238 17.02953454  9.73116259  0.          30.004418   1.62186043
  0.          2.43279065  0.81093022  4.8655813 ]
 [20.59741431  9.80829253  0.          6.86580477 30.40570684  0.
  0.          20.59741431  4.90414627  0.          ]
 [ 8.82746328  0.          4.90414627  1.96165851 19.61658506  0.
  0.98082925  0.          10.78912178  0.          ]
 [ 2.24164321  3.73607201  1.4944288   1.4944288   0.          13.44985923
 23.91086086  8.21935842 15.69150244  5.97771521]
 [ 0.          2.66190959  0.          0.          2.66190959  6.21112237
 10.64763834  3.54921278 23.95718627  3.54921278]
 [ 2.66190959  0.          0.          3.54921278  0.          14.19685112
 18.6333671   1.77460639 14.19685112 13.30954793]]
[[13.72 10.26  4.48  3.15 27.96 -0.48 -1.37  9.42  3.48  0.96]
 [16.64 12.48  5.43  3.96 33.62  0.59  0.19 11.76  5.96  1.91]
 [ 8.35  6.29  2.71  2.17 16.47  1.81  2.5   6.33  5.25  1.94]
 [ 2.18  1.9   0.58  2.07  1.04 12.77 20.21  5.14 19.76  8.44]
 [ 2.43  2.02  0.7   1.75  2.38  9.65 15.24  4.43 15.18  6.45]
 [ 1.2   1.15  0.27  1.72 -0.66 11.72 18.59  4.16 17.9   7.68]]
```

8) Latent semantic of terms는 U벡터와 S벡터의 곱으로 표현할 수 있고, Latent semantic of documents는 S벡터와 V transpose벡터의 곱으로 표현할 수 있다.

```
In [70]: cal_term = np.dot(U, S)
cal_doc = np.dot(S, VT)

print('term: ', cal_term)
print('documents: ', cal_doc)
```

```
term: [[-31.09650907 -15.57106884]
 [-39.01511766 -16.25003458]
 [-21.23757706 -4.72500638]
 [-19.09077616  26.63008721]
 [-16.23570969  19.30105351]
 [-15.6907574   25.2884699 ]]
documents: [[-21.90746992 -16.62485475 -7.047891   -6.37535895 -41.75980818
 -10.21357049 -15.26118074 -18.16095182 -21.97122062 -8.61190369]
 [-8.18610992 -5.76969664 -2.84391144  0.18367407 -21.13656132
 17.11437947 27.59235603 -0.85015879 23.10512026 10.29851345]]
```

9) Latent semantic of terms 및 documents를 데이터프레임의 형태로 변환



하여, term\_lsm, doc\_lsm라는 변수에 저장한다.

```
In [86]: term_lsm = pd.DataFrame(cal_term, index=df['Unnamed: 0'])
term_lsm.index.names = ['']

term_lsm
```

```
Out [86]:
```

	0	1
database	-31.096509	-15.571069
SQL	-39.015118	-16.250035
index	-21.237577	-4.725006
regression	-19.090776	26.630087
likelihood	-16.235710	19.301054
linear	-15.690757	25.288470

```
In [91]: doc_lsm = pd.DataFrame(cal_doc, columns=df.columns[1:])

doc_lsm
```

```
Out [91]:
```

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
0	-21.90747	-16.624855	-7.047891	-6.375359	-41.759808	-10.213570	-15.261181	-18.160952	-21.971221	-8.611904
1	-8.18611	-5.769697	-2.843911	0.183674	-21.136561	17.114379	27.592356	-0.850159	23.105120	10.298513

## 요구사항 1) 단어 'database'와 가장 유사한 단어 탐색

10) 코사인 유사도를 구해서 database와 가장 유사한 단어를 데이터프레임화하고, sort\_values를 사용하여 similarity가 높은 순부터 나열한다.

```
In [105]: def cos_sim(A, B):
return np.dot(A, B) / (np.linalg.norm(A) * np.linalg.norm(B))
```

```
In [108]: for i in range(len(term_lsm)):
print(i+1, "번째 term과 term 'database'의 cosin similarity: ",
cos_sim(term_lsm.iloc[i],term_lsm.iloc[0]))
```

```
1 번째 term과 term 'database'의 cosin similarity: 0.9999999999999998
2 번째 term과 term 'database'의 cosin similarity: 0.9975802836629114
3 번째 term과 term 'database'의 cosin similarity: 0.9700604744635943
4 번째 term과 term 'database'의 cosin similarity: 0.15708251340435264
5 번째 term과 term 'database'의 cosin similarity: 0.2329581273492023
6 번째 term과 term 'database'의 cosin similarity: 0.0909754091396296
```

```
In [114]: cos_sim_term_lst = []
for i in range(len(term_lsm)):
cos_sim_term_lst.append(cos_sim(term_lsm.iloc[i],term_lsm.iloc[0]))
```

```
In [127]: cos_sim_term_df = pd.DataFrame(cos_sim_term_lst, index=df['Unnamed: 0'])
cos_sim_term_df.index.names = ['']
cos_sim_term_df = cos_sim_term_df.rename(columns={0 : 'cosine similarity'})

cos_sim_term_df.sort_values('cosine similarity', ascending=False)
```

Out [127]:

	cosine similarity
database	1.000000
SQL	0.997580
index	0.970060
likelihood	0.232958
regression	0.157083
linear	0.090975

Conclusion: 'database' 단어와 가장 유사도가 높은 단어는 'SQL'이다. 코사인 유사도를 기준으로 유사성을 측정해보면, 0.997580의 유사도가 산출된다.

## 요구사항 2) 문서 'D6'와 가장 유사한 문서 탐색

11) 코사인 유사도를 구해서 Document 6과 가장 유사한 문서를 데이터프레임화하고, sort\_values를 사용하여 similarity가 높은 순부터 나열한다.

```
In [149]: for i in range(len(doc_lsm.columns)):
print(i+1, "번째 doc과 doc D6의 cosin similarity: ",
cos_sim(doc_lsm.iloc[:, i], doc_lsm.iloc[:, 5]))

1 번째 doc과 doc D6의 cosin similarity: 0.1794710410884455
2 번째 doc과 doc D6의 cosin similarity: 0.20259206878901942
3 번째 doc과 doc D6의 cosin similarity: 0.15390588728186597
4 번째 doc과 doc D6의 cosin similarity: 0.5369796834566384
5 번째 doc과 doc D6의 cosin similarity: 0.0694424192215988
6 번째 doc과 doc D6의 cosin similarity: 1.0000000000000002
7 번째 doc과 doc D6의 cosin similarity: 0.9994609877171637
8 번째 doc과 doc D6의 cosin similarity: 0.4717481732489627
9 번째 doc과 doc D6의 cosin similarity: 0.9754154939824525
10 번째 doc과 doc D6의 cosin similarity: 0.987481971384068
```

```
In [150]: cos_sim_doc_lst = []
for i in range(len(doc_lsm.columns)):
cos_sim_doc_lst.append(cos_sim(doc_lsm.iloc[:, i], doc_lsm.iloc[:, 5]))
```

```
In [152]: cos_sim_doc_df = pd.DataFrame(cos_sim_doc_lst, index=df.columns[1:] )
cos_sim_doc_df = cos_sim_doc_df.rename(columns={0 : 'cosine similarity'})

cos_sim_doc_df.sort_values('cosine similarity', ascending=False)
```

Out [152]:

	cosine similarity
D6	1.000000
D7	0.999461
D10	0.987482
D9	0.975415
D4	0.536980
D8	0.471748
D2	0.202592
D1	0.179471
D3	0.153906
D5	0.069442

Conclusion: 문서 D6과 가장 유사도가 높은 단어는 'D7'이다. 코사인 유사도를 기준으로 유사성을 측정해보면, 0.999461의 유사도가 산출된다.