Data Analytics

Assignment -LSA-

윤장혁 교수님

산업공학과

201811527

이영은

Week9

- 주어진 Term-document matrix(data_week9.csv)를 이용하여 LSA를 수행한 후, 다음 요구사항에 대한 분석 수행

  1) 단어 'database'와 가장 유사한 단어 탐색

  2) 문서 'D6'과 가장 유사한 문서 탐색

- 주의 사항
  - Term-document matrix를 TF-IDF matrix로 변환한 후 LSA를 수행할 것
    - TF(raw count), IDF(inverse document frequency smooth)
  - 2개의 singular values를 활용
  - Cosine 유사도를 활용

1) Term-Document matrix → TF-IDF matrix

csv파일 읽어오기

In [134]:
```
import pandas as pd
from math import log
import csv
import numpy as np
from numpy.linalg import svd
from numpy.linalg import norm

f = open('data_week9.csv','r', encoding = 'utf-8')

lines = []
rdr = csv.reader(f)
for line in rdr:
    lines.append(line)

vocab = []
for i in range(1,len(lines)):
    vocab.append(lines[i][0])

print(vocab)
col = lines[0][1:]
print(col)
```

```
['database', 'SQL', 'index', 'regression', 'likelihood', 'linear']
['D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9', 'D10']
```

실행 코드	코드 설명
import pandas as pd from math import log import csv import numpy as np from numpy.linalg import svd	필요한 library를 import 하였습니다. tf_idf를 구하기 위해 math와 log를, svd와 cosine similarity를 구하기 위하여 numpy를 import하였고, linalg의 svd와 norm 도 import 하였습니다.

from numpy.linalg import norm	
f = open('data_week9.csv','r', encoding = 'utf-8') lines = [] rdr = csv.reader(f) for line in rdr: lines.append(line)	9주차 과제로 제공받은 data_week9.csv를 읽어서 lines에 append 하였습니다.
vocab = [] for i in range(1,len(lines)): vocab.append(lines[i][0]) print(vocab) col = lines[0][1:] print(col)	후에 data frame을 만들고 그 columns 명 또는 index로 쓰기 위하여 lines Term만 저장하였습니다. 또한 document 명들을 col에 넣었습니다.

```
In [125]: lines

Out[125]: [['', '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']]
```

■ TF : DTM

```
In [129]: #TF : DTM
          result = []
          for i in range(1, len(lines)):
              result.append(lines[i][1:])

          tf = pd.DataFrame(result, columns = col, index = vocab)

          matA = []
          for i in range(0, 6):
              mat = []
              for j in range(0, 10):
                  mat.append(float(result[i][j]))
              matA.append(mat)

          matDT = np.transpose(matA)
          matA

Out[129]: [[11.0, 21.0, 12.0, 0.0, 37.0, 2.0, 0.0, 3.0, 1.0, 6.0],
           [21.0, 10.0, 0.0, 7.0, 31.0, 0.0, 0.0, 21.0, 5.0, 0.0],
           [9.0, 0.0, 5.0, 2.0, 20.0, 0.0, 1.0, 0.0, 11.0, 0.0],
           [3.0, 5.0, 2.0, 2.0, 0.0, 18.0, 32.0, 11.0, 21.0, 8.0],
           [0.0, 3.0, 0.0, 0.0, 3.0, 7.0, 12.0, 4.0, 27.0, 4.0],
           [3.0, 0.0, 0.0, 4.0, 0.0, 16.0, 21.0, 2.0, 16.0, 15.0]]
```

실행 코드	코드 설명

result = []
for i in range(1, len(lines)):
result.append(lines[i][1:])
 	
tf = pd.DataFrame(result, columns = col, index = vocab)
 	
matA = []
for i in range(0, 6):
mat = []
for j in range(0, 10):
mat.append(float(result[i][j]))
matA.append(mat)
 	
matA
	result 리스트를 만들어서 , 행이름과 열 이름을 제외한 값들만 2차원 리스트로 만들었습니다.
	
	이를 matA로 출력했습니다.

■ **IDF**

In [130]:
```
#IDF
sum_list = []

for i in range(0, 6):
    sum = 0
    for j in range(0, 10):
        if matA[i][j] == 0:
            sum += 0
        else :
            sum += 1
        j += 1
    sum_list.append(sum)

N = 10
idf = []
for i in range(0,6):
    idf_value = log(N/(sum_list[i]+1))
    idf.append(idf_value)

print(sum_list)
print(idf)
```

```
[8, 6, 6, 9, 7, 7]
[0.10536051565782635, 0.3566749439387324, 0.3566749439387324, 0.0, 0.22314355131420976, 0.22314355131420976]
```

실행 코드	코드 설명

#IDF
sum_list = []
 	
for i in range(0, 6):
  sum = 0
  for j in range(0, 10):
  if matA[i][j] == 0:
  sum += 0
  else :
  sum += 1
  j += 1
  sum_list.append(sum)
 	
N = 10
idf = []
for i in range(0,6):
  idf_value = log(N/(sum_list[i]+1))
  idf.append(idf_value)
 	
print(sum_list)
print(idf)
	단어가 몇 개의 Document에 출현하는지 더하기 위하여 Term이 Document에 출현한다면 +1을, 그렇지 않다면 0을 더해서 sum_list를 만들었습니다.
	
	idf를 구하기 위해 식을 만들었고, idf는 위에 첨부한 내용과 같습니다.
	
	[0.10536051565782635,
	0.3566749439387324,
	0.3566749439387324, 0.0,
	0.22314355131420976,
	0.22314355131420976]
	
	입니다. (Term 순서대로)

■ **TF – IDF**

```
In [136]: #TF-IDF
          TF_IDF = []
          for i in range(0, 6):
              tf = []
              for j in range(0, 10):
                  tf.append(matA[i][j]*idf[i])
              TF_IDF.append(tf)
          TF_IDF

          TF_IDF_df = pd.DataFrame(data = TF_IDF, columns = col, index = vocab)
          TF_IDF_df
```

실행 코드	코드 설명

#TF-IDF
TF_IDF = []
for i in range(0, 6):
  tf = []
	위에서 구한 IDF를 TF에 곱해주었습니다.
	
	그럼 다음과 같은 결과가 나오게 됩니다.

```
    for j in range(0, 10):
        tf.append(matA[i][j]*idf[i])
    TF_IDF.append(tf)
TF_IDF

TF_IDF_df = pd.DataFrame(data = TF_IDF,
columns = col, index = vocab)
TF_IDF_df
```

TF-IDF 결과

Out[136]:

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
database	1.158966	2.212571	1.264326	0.000000	3.898339	0.210721	0.000000	0.316082	0.105361	0.632163
SQL	7.490174	3.566749	0.000000	2.496725	11.056923	0.000000	0.000000	7.490174	1.783375	0.000000
index	3.210074	0.000000	1.783375	0.713350	7.133499	0.000000	0.356675	0.000000	3.923424	0.000000
regression	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
likelihood	0.000000	0.669431	0.000000	0.000000	0.669431	1.562005	2.677723	0.892574	6.024876	0.892574
linear	0.669431	0.000000	0.000000	0.892574	0.000000	3.570297	4.686015	0.446287	3.570297	3.347153

2) SVD in LSA

- 2개의 singular values를 활용

■ U, Sigma, V 구하기

- SVD를 실행하기 위해서 np.linalg.svd를 사용하였습니다.

In [145]:
```
U, Sigma, Vt = svd(TF_IDF, full_matrices = True)
print(U.shape, Sigma.shape, Vt.shape)
print(np.round(U,3))
print(np.round(Sigma,3))
print(np.round(Vt,3))
```

```
(6, 6) (6,) (10, 10)
[[-0.222  0.06  -0.187  0.537 -0.789  0.   ]
 [-0.856  0.249  0.423 -0.149  0.058 -0.   ]
 [-0.429 -0.134 -0.804  0.107  0.374 -0.   ]
 [ 0.     0.     0.    -0.    -0.    -1.   ]
 [-0.146 -0.62  -0.099 -0.64  -0.418  0.   ]
 [-0.108 -0.73   0.361  0.517  0.242  0.   ]]
[18.317  9.691  4.991  3.18   2.158  0.   ]
[[-0.443 -0.199 -0.057 -0.139 -0.737 -0.036 -0.057 -0.364 -0.246 -0.034]
 [ 0.105  0.063 -0.017 -0.013  0.167 -0.367 -0.529  0.104 -0.662 -0.305]
 [ 0.123  0.206 -0.335  0.161 -0.371  0.219  0.228  0.638 -0.346  0.201]
 [ 0.062  0.072  0.274  0.052  0.246  0.302  0.235 -0.405 -0.566  0.472]
 [ 0.409 -0.843 -0.153  0.291 -0.022  0.02   0.067 -0.037 -0.079 -0.03 ]
 [ 0.171 -0.062 -0.551 -0.613  0.103 -0.165 -0.142 -0.101  0.047  0.467]
 [-0.129 -0.045  0.034 -0.066  0.095 -0.693  0.681  0.064 -0.132 -0.038]
 [ 0.019 -0.31   0.564 -0.61  -0.052  0.161  0.012  0.424 -0.068 -0.014]
 [ 0.549  0.205  0.386  0.121 -0.435 -0.377 -0.157 -0.083  0.158  0.323]
 [-0.508 -0.233  0.12   0.317  0.139 -0.221 -0.31   0.29   0.079  0.564]]
```

차례대로 U, Sigma, Vt를 나타냅니다.

■ 2개의 singular value 활용

```
In [147]: newU = []
          for i in range(0, len(U)):
              mat = []
              for j in range(0, 2):
                  mat.append(U[i][j])
              newU.append(mat)
          newU
```

```
Out[147]: [[-0.22213979265053518, 0.05987532893970823],
           [-0.8564263590711704, 0.24898713357442606],
           [-0.42934884658924233, -0.13425722396973866],
           [0.0, 0.0],
           [-0.1458117878465881, -0.6196895111461869],
           [-0.10763965099694559, -0.729643926779421]]
```

```
In [146]: newS = [[18.371, 0], [0, 9.691]]
          newS
```

```
Out[146]: [[18.371, 0], [0, 9.691]]
```

```
In [148]: newVt = []
          for i in range(0, 2):
              newVt.append(Vt[i])
          newVt
```

```
Out[148]: [array([-0.4434508 , -0.19893205, -0.05713637, -0.13870511, -0.73680427,
                  -0.0359713 , -0.05721474, -0.36377733, -0.24557191, -0.03444204]),
           array([ 0.10473239,  0.06250476, -0.0168955 , -0.01293834,  0.16653961,
                  -0.36740293, -0.52899917,  0.10372154, -0.66197584, -0.30518974])]
```

- 2개의 singular value를 활용하기 위하여

  U = 6 x 2

  Sigma = 2 x 2

  Vt = 2 x 10 matrix로 차원을 축소했습니다.

■ Term – Latent Dimension matrix

```
In [163]: T_latent = np.dot(newU, newS)
          T_DF = pd.DataFrame(data = T_latent, columns = ['DIM1', 'DIM2'], index = vocab)
          T_DF
```

Out[163]:

	DIM1	DIM2
database	-4.080930	0.580252
SQL	-15.733409	2.412934
index	-7.887568	-1.301087
regression	0.000000	0.000000
likelihood	-2.678708	-6.005411
linear	-1.977448	-7.070979

차원을 축소한 U와 Sigma의 행렬곱으로 표현한 Term – Latent Dimension matrix는 위에 첨부한 내용과 같습니다.

■ **Document – Latent Dimension matrix**

```
In [172]: D_latent = np.dot(newS, newVt)
          D_latentT = np.transpose(D_latent)
          D_DF = pd.DataFrame(data = D_latentT, columns = ['DIM1','DIM2'], index = col)
          D_DF
          D_DF
```

Out[172]:

	DIM1	DIM2
D1	-8.146635	1.014962
D2	-3.654581	0.605734
D3	-1.049652	-0.163734
D4	-2.548151	-0.125385
D5	-13.535831	1.613935
D6	-0.660829	-3.560502
D7	-1.051092	-5.126531
D8	-6.682953	1.005165
D9	-4.511401	-6.415208
D10	-0.632735	-2.957594

차원을 축소한 Sigma와 Vt의 행렬곱으로 표현한 Document – Latent Dimension matrix는 위에 첨부한 내용과 같습니다. (cosine 유사도를 편하게 구하기 위하여 matrix를 전치하였습니다.)

3) Use cosine similarity

■ **단어 'database'와 가장 유사한 단어 탐색**

```
In [164]: #T-T
          def cos_sim(A, B):
              A = T_latent[A]
              B = T_latent[B]
              return np.dot(A, B)/(norm(A)*norm(B))

          print("database, SQL : ",cos_sim(0, 1))
          print("database, index : ",cos_sim(0, 2))
          print("database, regression : ",cos_sim(0, 3))
          print("database, likelihood : ",cos_sim(0, 4))
          print("database, linear : ",cos_sim(0, 5))
```

```
database, SQL :  0.9999401741885063
database, index :  0.9539305808583386
database, regression :  nan
database, likelihood :  0.27474461590361815
database, linear :  0.13107282724101718
```

- 단어 database와 가장 유사한 단어는 cosine similarity가 가장 높은 SQL입니다.

■ 문서 'D6'과 가장 유사한 문서 탐색

In [166]:
```
#D-D
def cos_sim(A, B):
    A = D_latentT[A]
    B = D_latentT[B]
    return np.dot(A, B)/(norm(A)*norm(B))
print("D6,D10 : ",cos_sim(5, 9))
print("D6,D9 : ",cos_sim(5, 8))
print("D6,D8 : ",cos_sim(5, 7))
print("D6,D7 : ",cos_sim(5, 6))
print("D6,D5 : ",cos_sim(5, 4))
print("D6,D4 : ",cos_sim(5, 3))
print("D6,D3 : ",cos_sim(5, 2))
print("D6,D2 : ",cos_sim(5, 1))
print("D6,D1 : ",cos_sim(5, 0))
```

```
D6,D10 :  0.9996288273450407
D6,D9 :  0.9092230532344926
D6,D8 :  0.03421673262344409
D6,D7 :  0.9998248696651012
D6,D5 :  0.06479229526393016
D6,D4 :  0.23058469847296884
D6,D3 :  0.3318403185197385
D6,D2 :  0.019257403225042845
D6,D1 :  0.05952858725741625
```

- 문서 D6와 가장 유사한 단어는 cosine similarity가 0.998로가장 높은 D7입니다.