한국어 임베딩 5장: 문장 수준의 임베딩

5-1 잠재의미 분석(LSA)

> LSA?

잠재 의미 분석(LSA)는 TF-IDF 행렬/ 단어-문맥 행렬/ PMI 행렬에 SVD로 차원축소를 시행하여, 임베딩을 만드는 방법이다.

TF-IDF는 단순히 단어의 빈도수로 구성된 행렬이기 때문에 단어의 의미를 고려하지 못한다는 단점이 있다. 이 문제를 해결한 방법이 잠재 의미 분석이다.

잠재 의미 분석을 하기 위해서는 Truncated SVD를 이용해야 한다

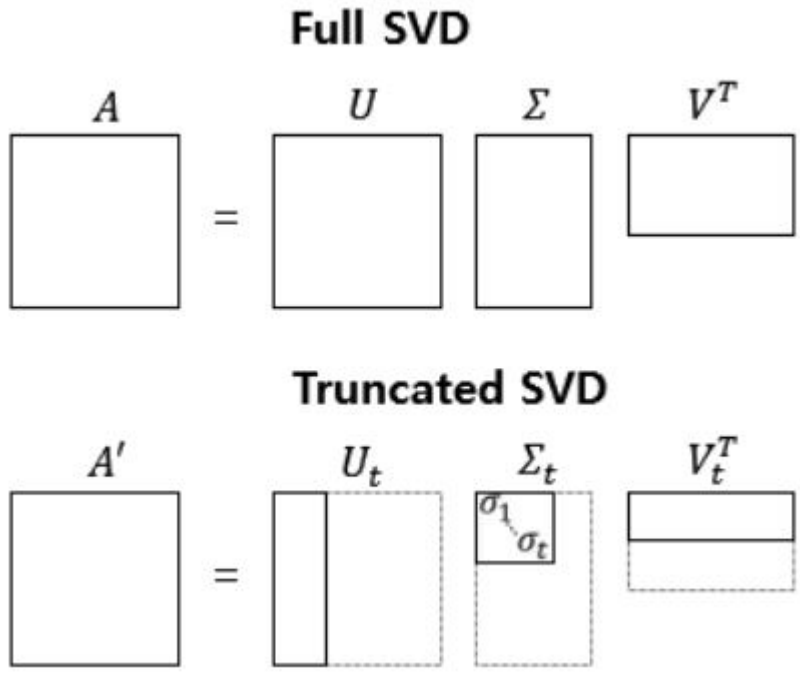
> SVD, Truncated SVD

SVD는 mxn 크기의 행렬을 $U, \sum, V$ 의 곱으로 분해하는 것이다

$$A = U \sum V^T$$

- U: mxm 직교행렬
- $\sum$: mxn 직사각 대각행렬
- V : nxn 직교행렬

LSA에서는 SVD에서 일부 벡터를 삭제하여 Truncated SVD를 활용한다. 절단된 SVD에서는 대각행렬 $\sum$에서 상위 t개의 특이값 $\sigma_1, \sigma_2, \ldots, \sigma_t$를 사용하고 나머지는 제거한다. 이에 따라 행렬 U와 V의 t열까지만 남겨 차원의 수를 맞춰준다.



> 행렬 분해로 이해하는 잠재 의미 분석

-	doc1	doc2	doc3
나	1	0	0
는	1	1	2
학교	1	1	0
에	1	1	0
가	1	1	0
ㄴ	1	0	0
다	1	0	1
영희	0	1	1
좋	0	0	1

이 단어-문서 행렬에 잠재 의미 분석을 수행해 보자. 우선 SVD를 수행하면 아래와 같다

$$A = U\Sigma V^T$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.17 & 0.27 & -0.40 \\ -0.63 & -0.41 & -0.03 \\ -0.32 & 0.37 & 0.21 \\ -0.32 & 0.37 & 0.21 \\ -0.32 & 0.37 & 0.21 \\ -0.17 & 0.27 & -0.40 \\ -0.33 & -0.12 & -0.52 \\ -0.30 & -0.29 & 0.49 \\ -0.15 & -0.39 & -0.13 \end{bmatrix} \begin{bmatrix} 3.61 & 0 & 0 \\ 0 & 2.04 & 0 \\ 0 & 0 & 1.34 \end{bmatrix} \begin{bmatrix} -0.63 & -0.53 & -0.57 \\ 0.56 & 0.20 & -0.80 \\ -0.54 & 0.83 & -0.17 \end{bmatrix}$$

9개의 단어, 3개의 문서로 이루어진 단어-문서 행렬에 절단된 SVD를 수행, 즉 LSA를 시행한다고 가정해보자. 그러면 U는 단어 임베딩, V는 문서 임베딩에 대응한다. 특이값을 2개만 남기고 제거해 Truncated SVD를 수행하면 아래와 같다

$$A' = U_2 \Sigma_2 V_2^T$$

$$\begin{bmatrix} 0.71 & 0.44 & -0.09 \\ 0.97 & 1.04 & 1.99 \\ 1.15 & 0.76 & 0.04 \\ 1.15 & 0.76 & 0.04 \\ 1.15 & 0.76 & 0.04 \\ 0.71 & 0.45 & -0.09 \\ 0.62 & 0.58 & 0.88 \\ 0.36 & 0.45 & 1.11 \\ -0.09 & 0.14 & 0.97 \end{bmatrix} = \begin{bmatrix} -0.17 & 0.27 \\ -0.63 & -0.41 \\ -0.32 & 0.37 \\ -0.32 & 0.37 \\ -0.32 & 0.37 \\ -0.17 & 0.27 \\ -0.33 & -0.12 \\ -0.30 & -0.29 \\ -0.15 & -0.39 \end{bmatrix} \begin{bmatrix} 3.61 & 0 \\ 0 & 2.04 \end{bmatrix} \begin{bmatrix} -0.63 & -0.53 & -0.57 \\ 0.56 & 0.20 & -0.80 \end{bmatrix}$$

단어 벡터들과 문서 벡터를 모두 2차원으로 표현할 수 있음을 확인 가능하다. 이로써 단어와 문맥간의 내재적인 의미를 효과적으로 보존해 문서 간 유사도 측정 등 모델의 성능 향상에 도움된다고 한다.

> 장,단점

truncated SVD를 진행하는 경우 전체 SVD를 하였을 때보다 계산비용이 낮아지며, 중요하지 않은 정보를 삭제하는 효과가 있다. 이는 설명력이 낮은 정보를 삭제하고 핵심적인 정보만 남긴다는 의미이다(노이즈 제거)

반면 새로운 데이터가 추가되면 처음부터 행렬을 다시 구성해 계산해야 한다는 단점이 있다

> 실습 - DTM 행렬로 절단된 SVD 만들어보기

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

In [1]:
```
#TF-IDF 행렬 구축
import numpy as np
A=np.array([[0,0,0,1,0,1,1,0,0],[0,0,0,1,1,0,1,0,0],[0,1,1,0,2,0,0,0,0],[1,0,0,0,0,0,0,1,1
]])
A
```

Out[1]:
```
array([[0, 0, 0, 1, 0, 1, 1, 0, 0],
       [0, 0, 0, 1, 1, 0, 1, 0, 0],
       [0, 1, 1, 0, 2, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 1, 1]])
```

In [2]:
```
#SVD분해
U, s, VT = np.linalg.svd(A, full_matrices = True)
```

In [3]:
```
#행렬의 크기 확인
print('Shape of A matrix:',np.shape(A))
print('Shape of U matrix:',np.shape(U))
print('Shape of s matrix:',np.shape(s))
print('Shape of VT matrix:',np.shape(VT))
```

```
Shape of A matrix: (4, 9)
Shape of U matrix: (4, 4)
Shape of s matrix: (4,)
Shape of VT matrix: (9, 9)
```

linalg.svd()를 실행하는 경우에 특이값 분해 결과의 대각행렬을 형성하는 것이 아니라, 특이값의 리스트를 반환한다. 따라서 s가 4x9 크기의 행렬이 아닌, 4개의 원소를 갖는 리스트가 나오는 것이다. 각각의 특이값을 대각원소로 갖는 행렬을 구축해야 한다.

In [4]:
```
s
```

Out[4]:
```
array([2.68731789, 2.04508425, 1.73205081, 0.77197992])
```

In [5]:
```
#각각의 특이값을 대각원소로 갖는 행렬로 구축
S = np.zeros((4, 9)) # 대각 행렬의 크기인 4 x 9의 임의의 행렬 생성
S[:4, :4] = np.diag(s) # 특이값을 대각행렬에 삽입
print(S.round(2))
print('Shape of S matrix:',np.shape(S))
```

```
[[2.69 0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   2.05 0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   1.73 0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.77 0.   0.   0.   0.   0.  ]]
Shape of S matrix: (4, 9)
```

이제 상위 2개의 σ_1, σ_2만 남기고 나머지는 제거하는 방식으로 truncated SVD를 진행하려 한다. 이는 전체 문서를 2개의 토픽으로 압축한다는 것을 의미한다. 차원을 맞추기 위해 행렬 U, V에 대해서도 2개의 열만 남긴다

In [6]:
```
# Truncated SVD
S=S[:2,:2]
U=U[:,:2]
VT=VT[:2,:]

A_prime=np.dot(np.dot(U,S), VT)
print('기존 행렬Wn',A)
print('Wn절단된 SVD 실행 후Wn',A_prime.round(2))
```

```
기존 행렬
 [[0 0 0 1 0 1 1 0 0]
 [0 0 0 1 1 0 1 0 0]
 [0 1 1 0 2 0 0 0 0]
 [1 0 0 0 0 0 0 1 1]]

절단된 SVD 실행 후
 [[ 0.   -0.17 -0.17  1.08  0.12  0.62  1.08 -0.   -0.  ]
 [ 0.    0.2   0.2   0.91  0.86  0.45  0.91  0.    0.  ]
 [ 0.    0.93  0.93  0.03  2.05 -0.17  0.03  0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.  ]]
```

In [7]:
```
print('Wn절단된 SVD 실행 후 U 행렬Wn',U.round(2))
print('Wn절단된 SVD 실행 후 VT 행렬Wn',VT.round(2))
```

```
절단된 SVD 실행 후 U 행렬
 [[ 0.24  0.75]
 [ 0.51  0.44]
 [ 0.83 -0.49]
 [ 0.   -0.  ]]

절단된 SVD 실행 후 VT 행렬
 [[ 0.    0.31  0.31  0.28  0.8   0.09  0.28  0.    0.  ]
 [ 0.   -0.24 -0.24  0.58 -0.26  0.37  0.58 -0.   -0.  ]]
```

U행렬은 4X9 크기에서 4X2로 축소되었고, V행렬 또한 9X9 크기에서 9X2로 축소되었다.

$$U = 문서의개수 X 토픽의수$$
$$V = 단어의개수 X 토픽의수$$

이렇게 구축된 문서벡터들과 단어벡터들을 통해서 문서 간의 유사도, 단어 간의 유사도 등을 구하는 것이 가능해진다.

In [8]:
```
# 문서 유사도 계산 예시
from numpy import dot
from numpy.linalg import norm
import numpy as np
def cos_sim(A, B):
       return dot(A, B)/(norm(A)*norm(B))
```

In [9]:
```
doc1 = U[0,]
doc2 = U[1,]
doc3 = U[2,]
doc4 = U[3,]
```

In [10]:
```
print(round(cos_sim(doc1, doc2),2))
print(round(cos_sim(doc1, doc3),2))
print(round(cos_sim(doc1, doc4),2))
```

```
0.86
-0.22
0.1
```

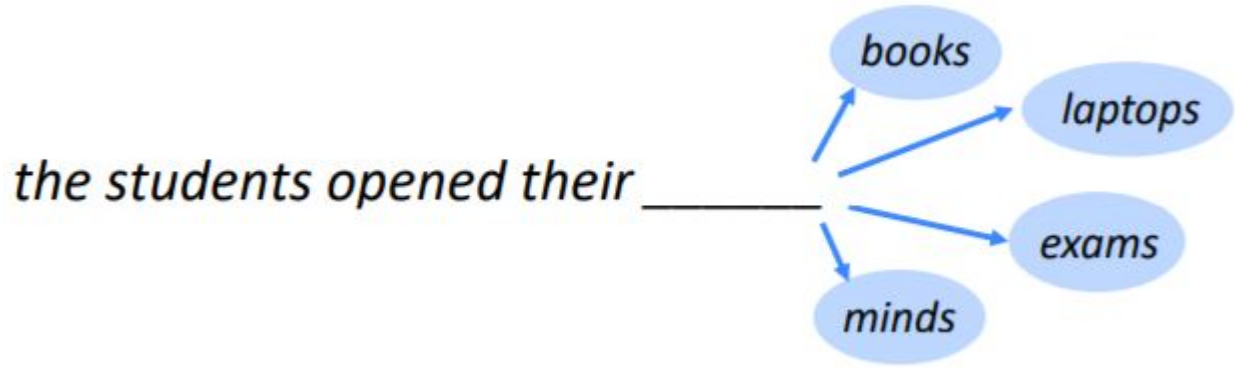
문서1과 가장 유사도가 높은 문서는 문서2임을 확인할 수 있다

5-2 Doc2vec

Doc2vec?

Doc2vec 모델은 문서 임베딩 기법 중에 하나다. Doc2vec은 언어모델을 기반으로 해서 문서 임베딩을 학습하는 모델이다. 우선 word matrix를 input으로 받는 언어모델에 대해 알아보자

언어 모델 기반 단어벡터 학습



언어 모델은 문장에서 단어들이 주어졌을 때 다음에 나올 단어를 맞추는 모델이다. 즉 $x^{(1)}, \ldots, x^{(t)}$ 이 주어졌을 때 $x^{(t+1)}$ 가 나올 조건부 확률을 최대화하는 것이 과제다

$$P(x^{(t+1)}|x^{(t)}, \ldots, x^{(1)})$$

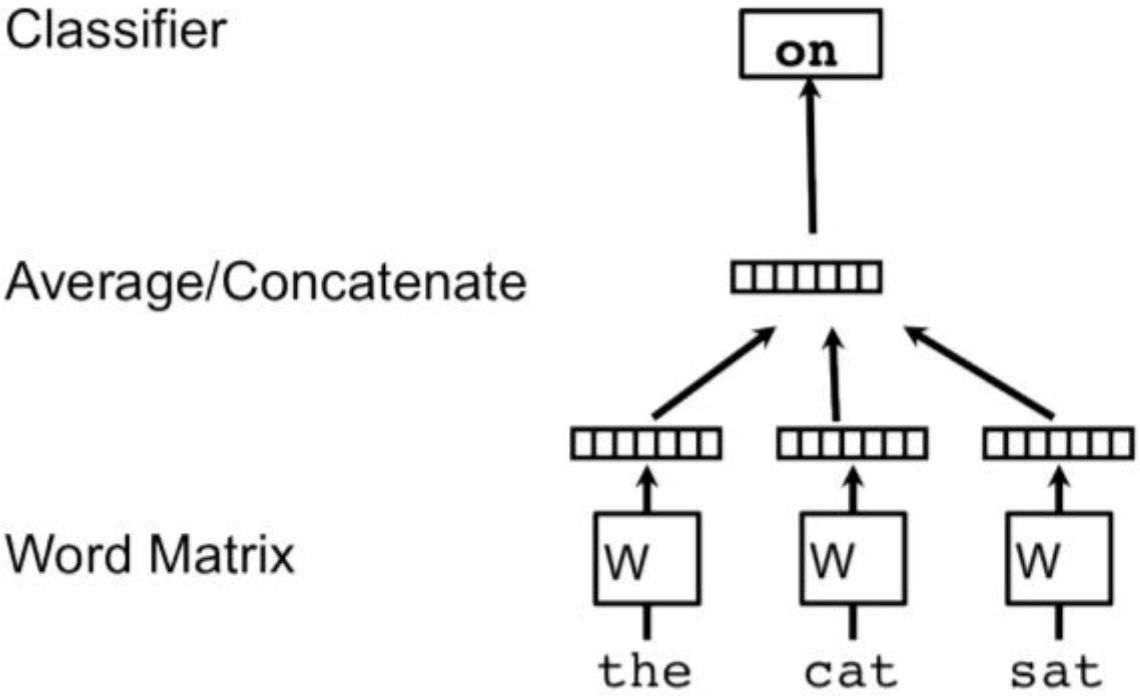
이 때 doc2vec은 앞의 모든 단어가 아닌 K개의 단어만 조건에 넣으며, 로그 확률의 평균을 최대화하는 방식으로 학습을 진행한다

$$L = \frac{1}{T}\sum_{t=k}^{T-1} logP(x^{(t)}|x^{(t-k)}, \ldots, x^{(t-1)})$$

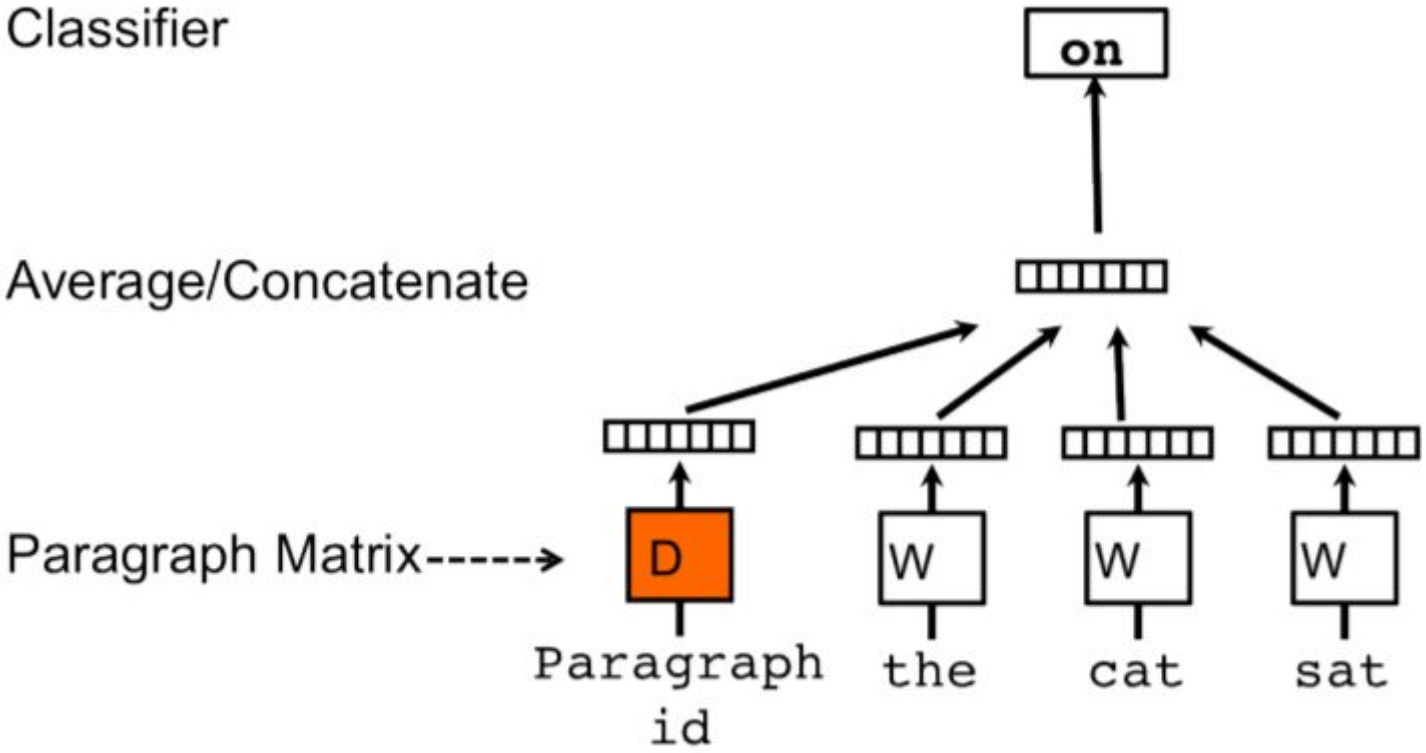
$$P(x^{(t)}|x^{(t-k)}, \ldots, x^{(t-1)}) = \frac{exp(y_{y_t})}{\sum exp(y_{y_i})}$$

$$y = b + Uh(w_{t-k}, \ldots, w_{t-1}, W)$$

y벡터는 단어행렬 W에서 참조한 K개의 단어 벡터를 평균을 취하거나 Concat하여 U라는 행렬과 내적해 bias term을 더해 구한다. 조건부 확률은 y를 소프트맥수 함수에 넣은 값이다.모델을 도식화하면 다음과 같다



DOC2vec PV-DM



PV-DM 모델은 기존의 방식에서 문서 ID를 input에 포함한 방식이다. 즉 k개의 단어들과 문서 ID를 넣어서 다음 단어를 예측한다는 뜻이다. 다른 과정은 완전히 동일하다.

이렇게 구성된 문서 임베딩은 해당 문서의 주제 정보를 함축한다고 한다. 문서 임베딩이 문서 내의 단어들과 함께 학습되며, 등장 순서 또한 고려하고 있기 때문이다.

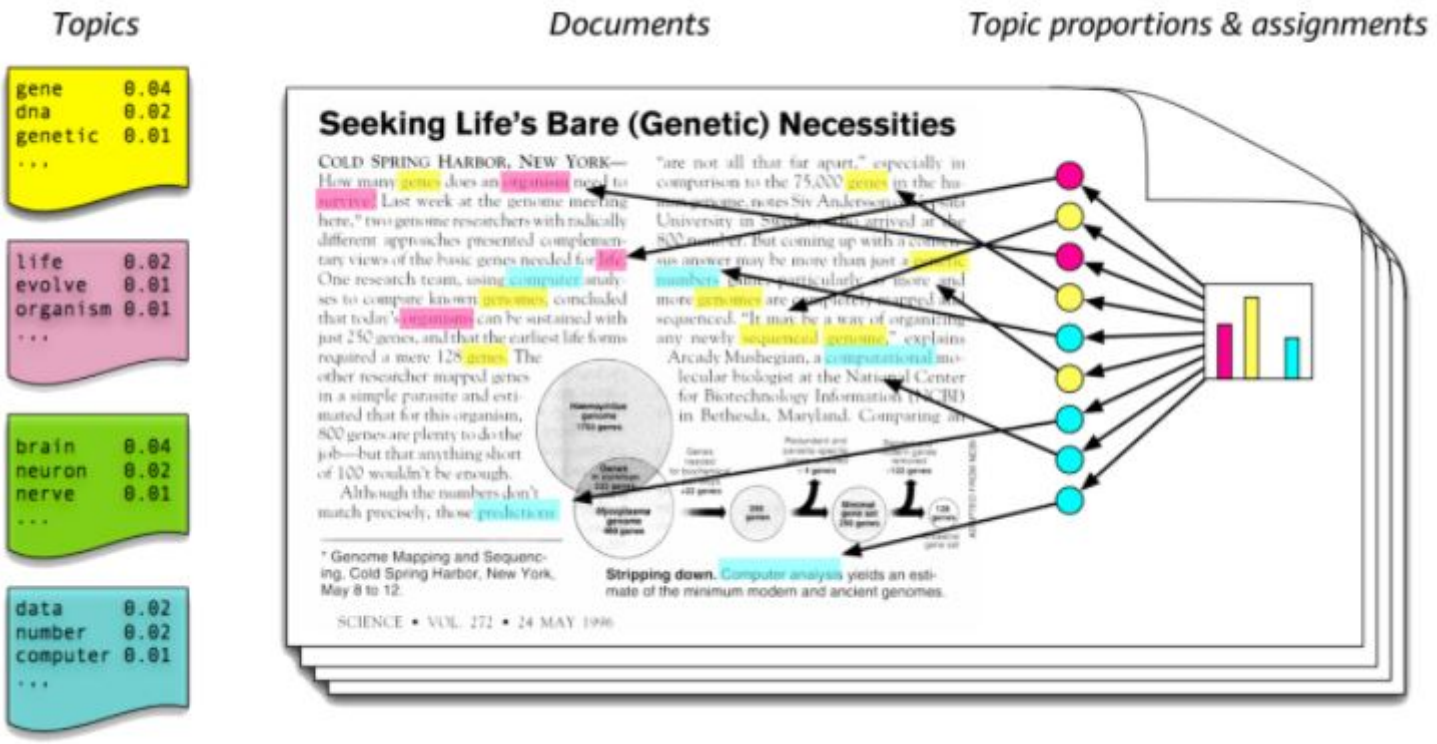
'the cat sat on the mat'이라는 문장이 있고 k=3이라고 할 때 학습 데이터는 <paragraph1,the,cat,sat - on> <paragraph1,cat,sat,on - the> <paragraph1,sat,on,the - mat> 등이 된다

5-3 잠재 디리클레 할당(LDA)

LDA란?

LDA는 주어진 문서에 대해서 각 문서에 어떤 토픽들이 존재하는지에 대한 확률 모형이다. 잠재된 토픽을 추출한다는 의미에서 **토픽모델링**이라고도 한다.

LDA 개요



우선 LDA는 특정 토픽에 특정 단어가 나타날 확률을 내어 준다. 예컨대 위 그림에서 노란색 토픽엔 gene이라는 단어가 등장할 확률이 0.04, dna는 0.02, genetic은 0.01이다. 또한 해당 문서는 노란색 토픽에 해당하는 단어가 많은 것으로 보아, 문서의 주제가 노란색 토픽일 가능성이 크다.

즉 LDA는
1) 문서들은 토픽들의 혼합으로 구성되어있다
2) 토픽들은 확률분포에 기반해 단어들을 생성한다
이 두가지를 가정하고 있다

LDA 결과물 예시

LDA는 문서의 토픽에 대한 확률 분포와, 각 토픽 내의 단어 분포를 추정한다.

문서1 : 저는 사과랑 바나나를 먹어요
문서2 : 우리는 귀여운 강아지가 좋아요
문서3 : 저의 깜찍하고 귀여운 강아지가 바나나를 먹어요

<각 문서의 토픽 분포>
문서1 : 토픽 A 100%
문서2 : 토픽 B 100%
문서3 : 토픽 B 60%, 토픽 A 40%

<각 토픽의 단어 분포>
토픽A : 사과 20%, 바나나 40%, 먹어요 40%, 귀여운 0%, 강아지 0%, 깜찍하고 0%, 좋아요 0%
토픽B : 사과 0%, 바나나 0%, 먹어요 0%, 귀여운 33%, 강아지 33%, 깜찍하고 16%, 좋아요 16%

> LDA 과정

1. 사전에 토픽에 개수 k개를 설정한다

k개의 토픽이 M개의 전체 문서에 걸쳐 분포되어 있다고 가정한다

1. 모든 단어를 k개 중 하나의 토픽에 할당한다

모든 문서의 모든 단어에 대해 k개 중 하나의 토픽을 랜덤으로 할당한다. 이 작업이 끝나면 각 문서의 토픽 분포와, 토픽 별 단어 분포를 갖는 상태이다.

1. 어떤 문서에서 w단어를 제외한 다른 단어들은 전부 올바른 토픽에 할당되어 있다는 가정 아래 w단어를 아래 두 기준에 따라 토픽에 재할당한다

기준1: 문서 d의 단어들 중 토픽 t에 해당하는 단어들의 비율
기준2: 단어 w를 갖고 있는 모든 문서들 중 토픽 t가 할당된 비율

doc1

word	apple	banana	apple	dog	dog
topic	B	B	???	A	A

doc2

word	cute	book	king	apple	apple
topic	B	B	B	B	B

두개의 문서 doc1 doc2에서 doc1의 세번째 단어 apple의 토픽을 결정한다고 하자. 해당 단어를 제외한 다른 단어들은 올바른 토픽에 할당되어 있다고 가정한다.

기준1

doc1

word	apple	banana	apple	dog	dog
topic	B	B	???	A	A

doc2

word	cute	book	king	apple	apple
topic	B	B	B	B	B

doc1의 단어들이 어떤 토픽에 해당하는지를 살펴보면 토픽 A와 토픽 B에 1/2의 비율로 할당되어 있기 때문에 어디에도 속할 가능성이 있다

기준2

doc1

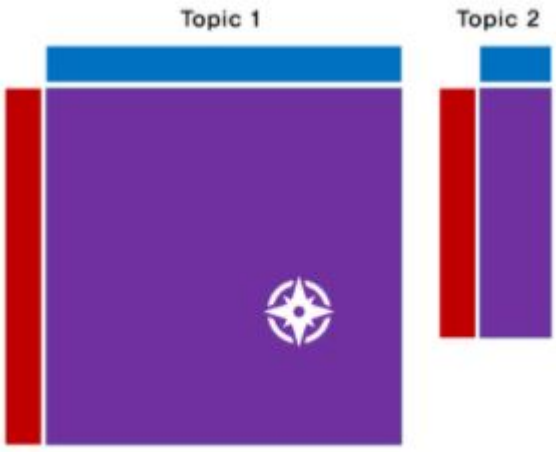
word	apple	banana	apple	dog	dog
topic	B	B	???	A	A

doc2

word	cute	book	king	apple	apple
topic	B	B	B	B	B

단어 apple이 문서 집합에서 어떤 토픽에 할당되었는지 살펴보자. 이 기준에 따르면 단어 apple은 토픽 B에 할당될 가능성이 있다.

최종적으로 두가지 기준에서 도출된 각 토픽에 대한 분포 A B의 곱을 기준으로 apple을 어떤 토픽에 할당할지 결정한다,



1. 토픽에 할당하는 작업이 일정한 값에 수렴할 때 까지 3을 반복한다

참고문헌

딥러닝을 이용한 자연언어처리 (https://wikidocs.net/24949)
ratsgo (https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/04/06/pcasvdlsa/)
Distributed Representations of Sentences and Documents (https://cs.stanford.edu/~quocle/paragraph_vector.pdf)