토픽 모델링(Topic Modeling) : 잠재 의미 분석(Latent Semantic Analysis)

LSA는 정확히 토필 모델링을 위해 최적화된 알고리즘은 아니지만, 토픽 모델링이라는 분야에 아이디어를 제공한 알고리즘.

Bow에 기반한 DTM이나 TF-IDF는 기본적으로 단어의 빈도 수를 이용한 수치화 방법이기 때문에 단어의 의미를 고려하지 못하지만, **LSA는 잠재된(Latent) 의미를 이끌어낼 수 있음**. LSA는 알기 위해서는 특이값 분해인 SVD를 알아야 함.

특이값 분해(Singular Value Decomposition, SVD)

- 특이값 분해란 : A가 $m \times n$ 행렬일 때, 다음과 같이 3개의 행렬의 곱으로 분해(decomposition)하는 것

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

  U : $m \times m$ 직교행렬

  V : $n \times n$ 직교행렬

  Σ : $m \times n$ 직사각 대각행렬

  이 때, Σ의 주대각원소를 특이값(singular value)이라고 함. 이때 SVD를 통해 나온 대각 행렬 Σ는 내림차순으로 정렬되어 있다는 특징을 가짐.

  $\Sigma = \begin{bmatrix} a_1 & 0 & 0 & 0 \\ 0 & a_2 & 0 & 0 \\ 0 & 0 & a_3 & 0 \end{bmatrix}$

  즉, 위 Σ의 주대각원소 a_1, a_2, a_3은 내림차순으로 정렬되어 있음

LSA 의미

American food인 pizza, pizza humberger cookie, hamburger와

Japanese food인 ramen, shshi, ramen shshi 메뉴가 있는 경우

- DTM인 경우 ← 단어기반

	pizza	hamburger	cookie	ramen	sushi

	pizza	hamburger	cookie	ramen	sushi
pizza	1	0	0	0	0
pizza hamburger cookie	1	1	1	0	0
hamburger	0	1	0	0	0
ramen	0	0	0	1	0
sushi	0	0	0	0	1
ramen sushi	0	0	0	1	1

cosine similarity(pizza, hamburger) = 0

cosine similarity(pizza, ramen) = 0

- LSA인 경우 ← 단어기반
  DTM이나 TF-IDF 행렬에 절단된 SVD를 사용하여 차원을 축소시키고, 단어들의 잠재적인 의미를 이끌어낸다는 아이디어를 가지고 있음
  - A는 word-document matrix

A =

	Pizza(d1)	pizza hamburger cookie(d2)	hamburger (d3)	ramen (d4)	sushi (d5)	ramen sushi (d6)
pizza(w1)	1	1	0	0	0	0
hamburger (w1)	0	1	1	0	0	0
cookie(w3)	0	1	0	0	0	0
ramen(w4)	0	0	0	1	0	1
sushi(w5)	0	0	0	0	1	1

  - 특이값 분해

$A \approx U \times \Sigma \times V^T$

U

	t1	t2	t3	t4	t5
w1	0.6	0	0	0.7	-0.3
w2	0.6	0	0	-0.7	-0.3
w3	0.5	0	0	0	0.9
w4	0	0.7	-0.7	0	0
w5	0	0.7	0.7	0	0

Word matrix for topic

Σ

	t1	t2	t3	t4	t5	t6
t1	1.9	0	0	0	0	0
t2	0	1.7	0	0	0	0
t3	0	0	1	0	0	0
t4	0	0	0	1	0	0
t5	0	0	0	0	0.5	0

Topic Strength

V^T

	d1	d2	d3	d4	d5	d6
t1	0.3	0.9	0.3	0	0	0
t2	0	1.7	0	0.4	0.4	0.8
t3	0	0	0	-0.7	0.7	0
t4	0.7	0	-0.7	0	0	0
t5	-0.6	0.5	-0.6	0	0	0
t6	0	0	0	-0.6	-0.6	0.6

Document matrix for topic

  - Document vector
    원하는 topic은 2개이기에 대각 행렬 Σ의 대각원소의 값 중 상위 2개만을 가지는 절단된 SVD를 사용하여 차원이 2개로 감소된 Document vector를 만듦.
    즉, Document vector = $\Sigma(topic\ stength) \times V(Document\ matrix\ for\ topic)$

Σ

	t1	t2
t1	1.9	0
t2	0	1.7

$\times$

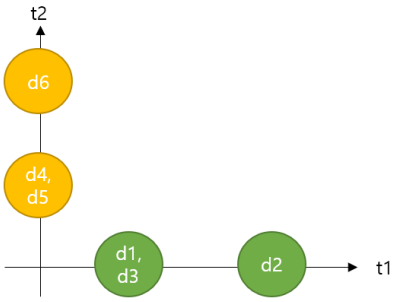
V^T

	d1	d2	d3	d4	d5	d6
t1	0.3	0.9	0.3	0	0	0
t2	0	1.7	0	0.4	0.4	0.8

$=$

$document\ vector$

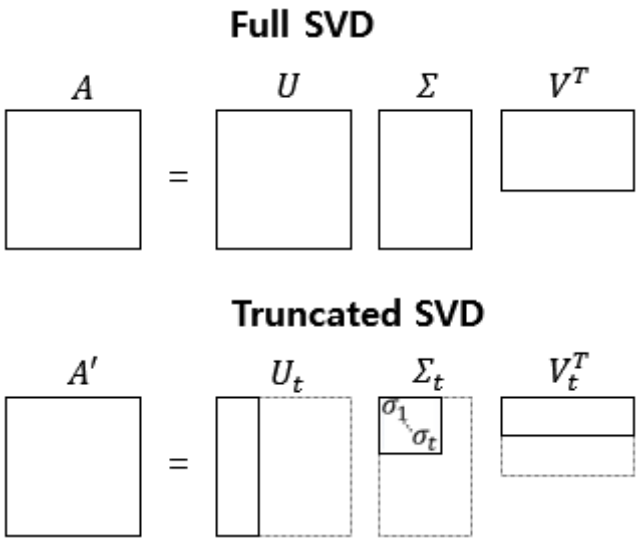
	d1	d2	d3	d4	d5	d6
t1	0.57	1.71	0.57	0	0	0
t2	0	0	0	0.68	0.68	1.36



cosine similarity(pizza, hamburger) = 1

cosine similarity(pizza, ramen) = 0

절단된 SVD(Truncated SVD)



- 절단된 SVD : 대각 행렬 Σ의 대각 원소의 값 중에서 상위값 t개에 해당하는 벡터들을 남겨두고 나머지 벡터들을 삭제시킨 것.
- 절단된 SVD를 수행하면 값의 손실이 일어나, 기존의 행렬 A를 복수할 수 없음.
- t를 크게 작으면 기존의 행렬 A로부터 다양한 의미를 가져갈 수 있지만, t를 작게 잡아야만 노이즈를 제거할 수 있음
- 데이터의 차원을 줄이는 이유 : 계산 비용이 낮아지는 효과, **상대적으로 중요하지 않은 정보를 삭제하는 효과**. → 기존 행렬에서는 드러나지 않았던 **심층적인 의미를 알게 해줌**.

잠재 의미 분석 실습

-	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

```
import numpy as np
# A는 DTM
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]])
np.shape(A)
```

```
(4,9)
```

```
# s == Σ
U, s, VT = np.linalg.svd(A, full_matrices = True)
```

```
# U (round → 반올림 함수)
print(U.round(2))
np.shape(U)
```

```
[[-0.24  0.75  0.   -0.62]
 [-0.51  0.44 -0.    0.74]
 [-0.83 -0.49 -0.   -0.27]
 [-0.   -0.    1.    0.  ]]
(4, 4)
```

```
print(s.round(2))
np.shape(s)
```

```
# Numpy의 linalg.svd()는 특이값 분해의 결과로, 대각 행렬이 아니라 특이값의 리스트를 반환
[2.69 2.05 1.73 0.77]
(4,)
```

```
# 특이값의 리스트 → 대각행렬 Σ
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))
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.  ]]
(4, 9)
```

```
# VT
print(VT.round(2))
np.shape(VT)
```

```
[[-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.  ]
 [ 0.58 -0.    0.    0.   -0.    0.   -0.    0.58  0.58]
 [ 0.   -0.35 -0.35  0.16  0.25 -0.8   0.16 -0.   -0.  ]
 [-0.   -0.78 -0.01 -0.2   0.4   0.4  -0.2   0.    0.  ]
 [-0.29  0.31 -0.78 -0.24  0.23  0.23  0.01  0.14  0.14]
 [-0.29 -0.1   0.26 -0.59 -0.08 -0.08  0.66  0.14  0.14]
 [-0.5  -0.06  0.15  0.24 -0.05 -0.05 -0.19  0.75 -0.25]
 [-0.5  -0.06  0.15  0.24 -0.05 -0.05 -0.19 -0.25  0.75]]
(9, 9)
```

```
# U X S X VT == A 인지 아닌지 확인하기 (Numpy.allclose()는 2개의 행렬이 동일하면 True를, 아니면 False를 반환하는 함수)
np.allclose(A, np.dot(np.dot(U,S), VT).round(2))
```

절단된 SVD(Truncated SVD) 실행

```
S = S[:2,:2]
print(S.round(s))
U=U[:,:2]
print(U.round(2))
VT=VT[:2,:]
print(VT.round(2))
```

```
[[2.69 0.  ]
 [0.   2.05]]
[[-0.24  0.75]
 [-0.51  0.44]
 [-0.83 -0.49]
 [-0.   -0.  ]]
[[-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.  ]]
```

```
A_prime=np.dot(np.dot(U,S), VT)
print(A)
print(A_prime.round(2))
```

```
[[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]]
[[ 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.  ]]
# 대체적으로 기존에 0인 값들은 0에 가까운 값이 나오고, 1인 값들은 1에 가까운 값이 나오는 것을 볼 수 있음. 하지만 값이 제대로 복구되지 않은 구간(4행)도 존재함.
```

사이킷런의 Twenty Newsgroups 데이터를 이용한 실습

사이킷런에서는 Twenty Newsgroups이라고 불리는 20개의 다른 주제를 가진 뉴스 데이터 제공. 앞서 언급했듯이 LSA가 토픽 모델링에 최적화 된 알고리즘은 아니지만, 토픽 모델링이라는 분야의 시초가 되는 알고리즘. 이 실습에서는 **LSA를 사용해서 문서의 수를 원하는 토픽의 수로 압축한 뒤에 각 토픽당 가장 중요한 단어 5개를 출력하는 실습으로 토픽 모델링을 수행**.

- 뉴스데이터에 대한 이해

```
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
dataset = fetch_20newsgroups(shuffle=True, random_state=1, remove=
('headers', 'footers', 'quotes'))
documents = dataset.data
len(documents)
```

```
11314 # 즉 훈련에 사용할 뉴스는 총 11,314개
```

```
documents[1]
```

```
"\n\n\n\n\n\n\nYeah, do you expect people to read the FAQ, etc. and actually
accept hard\natheism?  No, you need a little leap of faith, Jimmy.  Your
logic runs out\nof steam!\n\n\n\n\n\n\n\nJim,\n\nSorry I can't pity you,
Jim.  And I'm sorry that you have these feelings of\ndenial about the faith
you need to get by.  Oh well, just pretend that it will\nall end happily
ever after anyway.  Maybe if you start a new newsgroup,\nalt.atheist.hard,
you won't be bummin' so much?\n\n\n\n\n\n\nBye-Bye, Big Jim.  Don't forget
your Flintstone's Chewables!  :) \n--\nBake Timmons, III"
```

```
print(dataset.target_names) # 데이터의 카테고리 확인
```

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x',
'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball',
'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space',
'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast',
'talk.politics.misc', 'talk.religion.misc']
```

- 텍스트 전처리

```
news_df = pd.DataFrame({'document':documents})
# 특수 문자 제거
news_df['clean_doc'] = news_df['document'].str.replace("[^a-zA-Z]", " ")
# 길이가 3이하인 단어는 제거 (길이가 짧은 단어 제거)
news_df['clean_doc'] = news_df['clean_doc'].apply(lambda x: ' '.join([w for
w in x.split() if len(w)>3]))
# 전체 단어에 대한 소문자 변환
news_df['clean_doc'] = news_df['clean_doc'].apply(lambda x: x.lower())
```

```
news_df['clean_doc'][1]
```

```
'yeah expect people read actually accept hard atheism need little leap faith jimmy your logic runs steam sorry pity sorry that have these feelings denial about faith need well just pretend that will happily ever after anyway maybe start newsgroup atheist hard bummin much forget your flintstone chewables bake timmons'
```

```
from nltk.corpus import stopwords
stop_words = stopwords.words('english') # NLTK로부터 불용어를 받아옵니다.
tokenized_doc = news_df['clean_doc'].apply(lambda x: x.split()) # 토큰화
tokenized_doc = tokenized_doc.apply(lambda x: [item for item in x if item not in stop_words])
# 불용어를 제거합니다. (our, about, just, that, will, after 단어들 제거)
```

```
print(tokenized_doc[1])
```

```
['yeah', 'expect', 'people', 'read', 'actually', 'accept', 'hard', 'atheism', 'need', 'little', 'leap', 'faith', 'jimmy', 'logic', 'runs', 'steam', 'sorry', 'pity', 'sorry', 'feelings', 'denial', 'faith', 'need', 'well', 'pretend', 'happily', 'ever', 'anyway', 'maybe', 'start', 'newsgroup', 'atheist', 'hard', 'bummin', 'much', 'forget', 'flintstone', 'chewables', 'bake', 'timmons']
```

- TF-IDF 행렬 만들기

```
# TfidfVectorizer 입력은 토큰화가 되지 않은 테스트 데이터이기에 역토큰화 수행
# 역토큰화 (토큰화 작업을 역으로 되돌림)
detokenized_doc = []
for i in range(len(news_df)):
    t = ' '.join(tokenized_doc[i])
    detokenized_doc.append(t)

news_df['clean_doc'] = detokenized_doc
```

```
news_df['clean_doc'][1]
```

```
'yeah expect people read actually accept hard atheism need little leap faith jimmy logic runs steam sorry pity sorry feelings denial faith need well pretend happily ever anyway maybe start newsgroup atheist hard bummin much forget flintstone chewables bake timmons'
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english',
max_features= 1000, # 상위 1,000개의 단어를 보존
max_df = 0.5,
smooth_idf=True)

X = vectorizer.fit_transform(news_df['clean_doc'])
X.shape # TF-IDF 행렬의 크기 확인
```

```
(11314, 1000)
```

- 토픽 모델링(Topic Modeling)

```
# 20개의 topic을 가졌다고 가정하고 토픽 모델링
from sklearn.decomposition import TruncatedSVD
svd_model = TruncatedSVD(n_components=20, algorithm='randomized',
n_iter=100, random_state=122)
svd_model.fit(X)
len(svd_model.components_)
```

```
20
```

```
# svd_model.componets_는 LSA에서 VT에 해당
np.shape(svd_model.components_)
```

```
(20, 1000) # 토픽 수 X 단어 집합의 크기
```

```
terms = vectorizer.get_feature_names() # 단어 집합. 1,000개의 단어가 저장됨.

def get_topics(components, feature_names, n=5):
    for idx, topic in enumerate(components):
        print("Topic %d:" % (idx+1), [(feature_names[i], topic[i].round(5))
for i in topic.argsort()[:-n - 1:-1]])
get_topics(svd_model.components_,terms)
```

```
# 각 20개의 행의 각 1,000개의 열 중 가장 값이 큰 5개의 값을 찾아서 단어로 출력
Topic 1: [('like', 0.2138), ('know', 0.20031), ('people', 0.19334),
('think', 0.17802), ('good', 0.15105)]
Topic 2: [('thanks', 0.32918), ('windows', 0.29093), ('card', 0.18016),
('drive', 0.1739), ('mail', 0.15131)]
Topic 3: [('game', 0.37159), ('team', 0.32533), ('year', 0.28205), ('games',
0.25416), ('season', 0.18464)]
Topic 4: [('drive', 0.52823), ('scsi', 0.20043), ('disk', 0.15518), ('hard',
0.15511), ('card', 0.14049)]
Topic 5: [('windows', 0.40544), ('file', 0.25619), ('window', 0.1806),
('files', 0.16196), ('program', 0.14009)]
Topic 6: [('government', 0.16085), ('chip', 0.16071), ('mail', 0.15626),
('space', 0.15047), ('information', 0.13582)]
Topic 7: [('like', 0.67121), ('bike', 0.14274), ('know', 0.11189), ('chip',
0.11043), ('sounds', 0.10389)]
Topic 8: [('card', 0.44948), ('sale', 0.21639), ('video', 0.21318),
('offer', 0.14896), ('monitor', 0.1487)]
Topic 9: [('know', 0.44869), ('card', 0.35699), ('chip', 0.17169), ('video',
0.15289), ('government', 0.15069)]
Topic 10: [('good', 0.41575), ('know', 0.23137), ('time', 0.18933), ('bike',
0.11317), ('jesus', 0.09421)]
Topic 11: [('think', 0.7832), ('chip', 0.10776), ('good', 0.10613),
('thanks', 0.08985), ('clipper', 0.07882)]
Topic 12: [('thanks', 0.37279), ('right', 0.21787), ('problem', 0.2172),
('good', 0.21405), ('bike', 0.2116)]
Topic 13: [('good', 0.36691), ('people', 0.33814), ('windows', 0.28286),
('know', 0.25238), ('file', 0.18193)]
Topic 14: [('space', 0.39894), ('think', 0.23279), ('know', 0.17956),
('nasa', 0.15218), ('problem', 0.12924)]
```

```
Topic 15: [('space', 0.3092), ('good', 0.30207), ('card', 0.21615),
('people', 0.20208), ('time', 0.15716)]
Topic 16: [('people', 0.46951), ('problem', 0.20879), ('window', 0.16),
('time', 0.13873), ('game', 0.13616)]
Topic 17: [('time', 0.3419), ('bike', 0.26896), ('right', 0.26208),
('windows', 0.19632), ('file', 0.19145)]
Topic 18: [('time', 0.60079), ('problem', 0.15209), ('file', 0.13856),
('think', 0.13025), ('israel', 0.10728)]
Topic 19: [('file', 0.4489), ('need', 0.25951), ('card', 0.1876), ('files',
0.17632), ('problem', 0.1491)]
Topic 20: [('problem', 0.32797), ('file', 0.26268), ('thanks', 0.23414),
('used', 0.19339), ('space', 0.13861)]
```

LSA의 장단점

- 장점 : 쉽고 빠른 구현. **단어의 잠재적인 의미를 이끌**어내어, 문서 유사도 계산 등에서 좋은 성능을 보임.
- 단점 : 계산된 LSA에 새로운 데이터를 추가하여 계산하려고 하면, 보통 처음부터 다시 계산해야함. 즉, 새**로운 정보에 대해 업데이트가 어렵** → 해결 : Word2Vec과 같은 단어의 의미를 벡터화할 수 있는 또 다른 방법인 인공신경망 기반 방법론 사용

출처

- https://wikidocs.net/24949
- https://www.youtube.com/watch?v=GVPTGq53H5I