



# 8장. 텍스트 분석 – Part2

1팀 방민지, 오지후, 함예린

# 목차

---

#01 토픽 모델링 – LSA, LDA

#02 문서 군집화

#03 문서 군집화 수행

#04 군집별 핵심 단어 추출하기

#05 문서 유사도

#06 한글 텍스트 처리



# 토픽 모델링



# #01 토픽 모델링 개요

✓ 토픽 모델링 문서 집합에 숨어 있는 주제를 찾아내는 것



# #01 토픽 모델링 개요

1. 많은 양의 텍스트 데이터를 분석할 때 머신러닝 기반의 토픽 모델링을 적용해 숨어 있는 중요 주제를 효과적으로 찾아낼 수 있음
2. 사람이 수행하는 토픽 모델링: 더 함축적인 의미로 문장을 요약  
ML 기반의 토픽 모델: 숨겨진 주제를 효과적으로 표현할 수 있는 중심 단어를 함축적으로 추출
3. ML 기반의 토픽 모델링에 자주 사용되는 기법: LSA, LDA

# #02-1 LSA (Latent Semantic Analysis) 개요

## ✓ BoW에 기반한 DTM이나 TF-IDF

: 단어의 빈도 수를 이용한 수치화 방법

➡ 단어의 의미를 고려하지 못한다는 단점

➡ DTM의 잠재된(Latent) 의미를 이끌어내는 방법으로 LSA 사용

## ✓ LSA

- 특이값 분해(SVD)를 이용하여 DTM의 차원을 축소해 문서 간, 단어 간의 잠재적 의미를 파악하는 기법
- DTM의 희소성을 줄이고, 단어들 간의 연관성을 기반으로 문서의 주제를 추출

# #02-2 SVD

## ✓ full SVD

: A가  $m \times n$  행렬일 때 다음과 같이 3개의 행렬의 곱으로 분해하는 것

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

$U : m \times m$  직교행렬 ( $AA^T = U(\Sigma \Sigma^T)U^T$ )

$V : n \times n$  직교행렬 ( $A^T A = V(\Sigma^T \Sigma)V^T$ )

$\Sigma : m \times n$  직사각 대각행렬

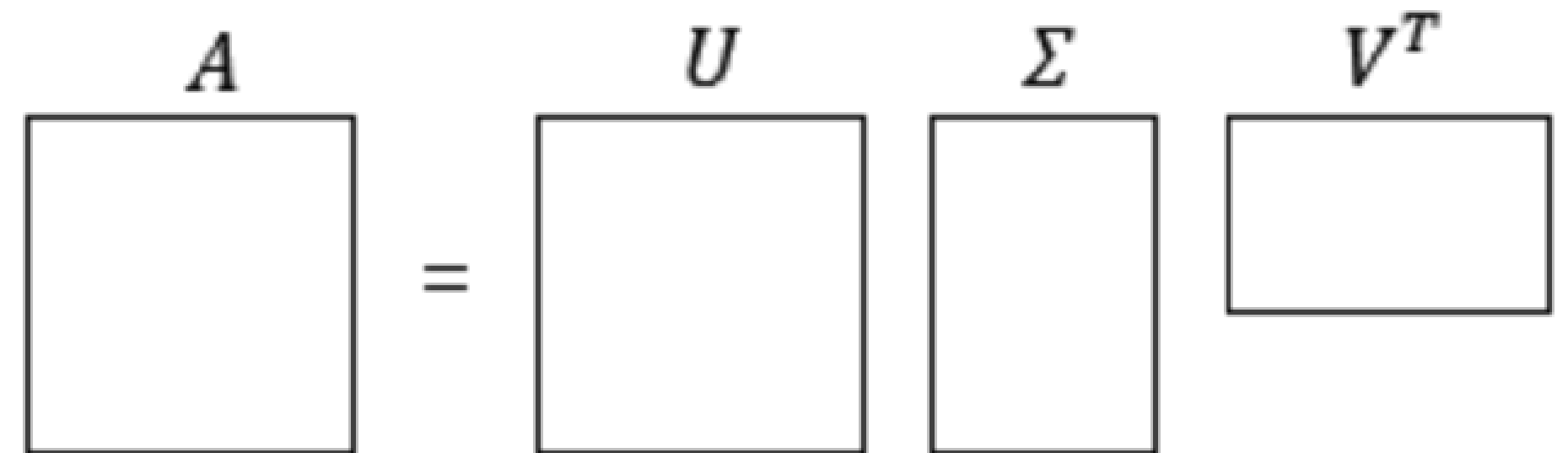
# #02-2 SVD

## ✓ 절단된 SVD(Truncated SVD)

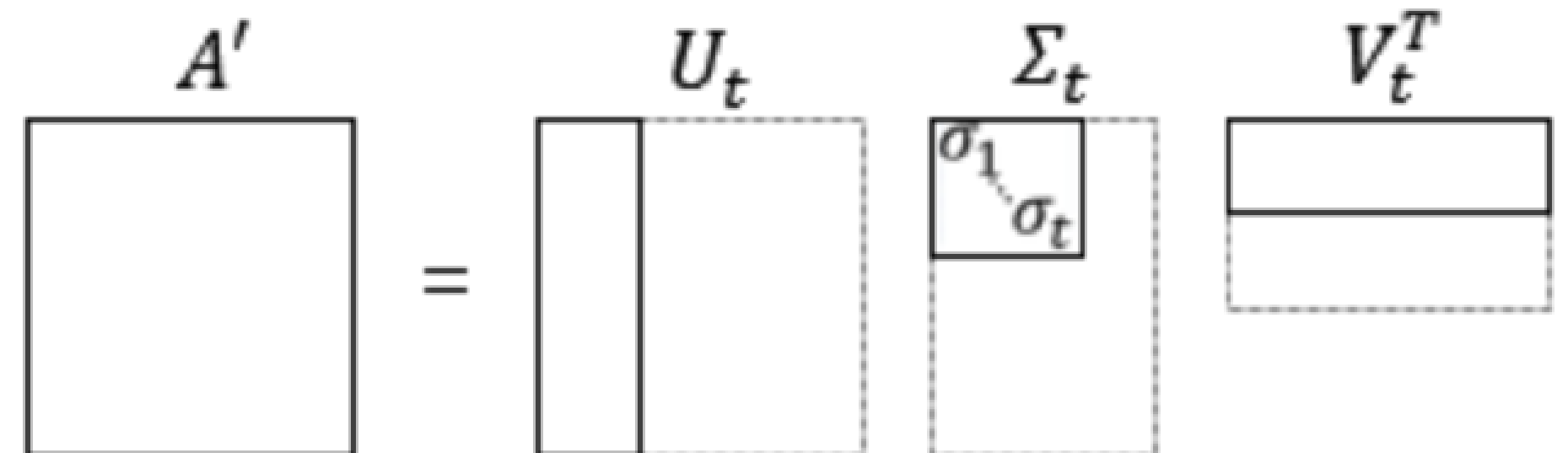
: 풀 SVD에서 나온 3개의 행렬에서  
일부 벡터들을 삭제

➡ LSA는 절단된 SVD를 사용

### Full SVD

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


### Truncated SVD

$$A' = U_t \Sigma_t V_t^T$$




# #02-2 SVD

## ✓ 절단된 SVD의 특성

1. 절단된 SVD는 대각 행렬  $\Sigma$ 의 대각 원소의 값 중에서 **상위값  $t$ 개만 남게 됨**

2. 절단된 SVD를 수행하면 **값의 손실**이 일어남

➡ 기존의 행렬  $A$ 를 **복구할 수 없음**, U행렬과 V행렬의  $t$ 열까지만 남김

- $t$ : 찾고자하는 토픽의 수를 반영한 하이퍼파라미터 값
- $t$ 를 크게 잡으면 기존의 행렬  $A$ 로부터 다양한 의미를 가져갈 수 있음
- $t$ 를 작게 잡아야 노이즈를 제거할 수 있음

# #02-2 SVD

## ✓ 절단된 SVD의 특성

3. 이렇게 일부 벡터들을 삭제하는 것을 **데이터의 차원을 줄인다고도** 말함

➡ 데이터의 차원을 줄이게되면 full SVD를 하였을 때보다 직관적으로 **계산 비용이 낮아지는 효과**를 얻을 수 있음

4. 상대적으로 중요하지 않은 정보를 삭제하는 효과를 갖고 있음

- 영상 처리 분야에서는 **노이즈를 제거**한다는 의미를 갖고 자연어 처리 분야에서는 **설명력이 낮은 정보를 삭제하고 설명력이 높은 정보를 남긴다는** 의미를 갖고 있음
- 기존의 행렬에서는 드러나지 않았던 심층적인 의미를 확인할 수 있음

# #02-3 LSA 장단점

## ✓ 장점

1. LSA는 쉽고 빠르게 구현이 가능
2. 단어의 잠재적인 의미를 이끌어낼 수 있어 문서의 유사도 계산 등에서 좋은 성능

## ✓ 단점

: SVD의 특성상 이미 계산된 LSA에 새로운 데이터를 추가하여 계산하려고 하면 보통 처음부터 다시 계산해야 함

➡ 새로운 정보에 대해 업데이트가 어려움.

➡ 최근 LSA 대신 Word2Vec 등 단어의 의미를 벡터화할 수 있는 또 다른 방법론인 인공 신경망 기반의 방법론이 각광받는 이유

# #03-1 LDA(Latent Dirichlet Allocation) 개요



## LDA

: 문서와 단어 간의 관계를 확률적 생성 모델로 설명하는 알고리즘

Topics

gene	0.04
dna	0.02
genetic	0.01
...	

life	0.02
evolve	0.01
organism	0.01
...	

brain	0.04
neuron	0.02
nerve	0.01
...	

data	0.02
number	0.02
computer	0.01
...	

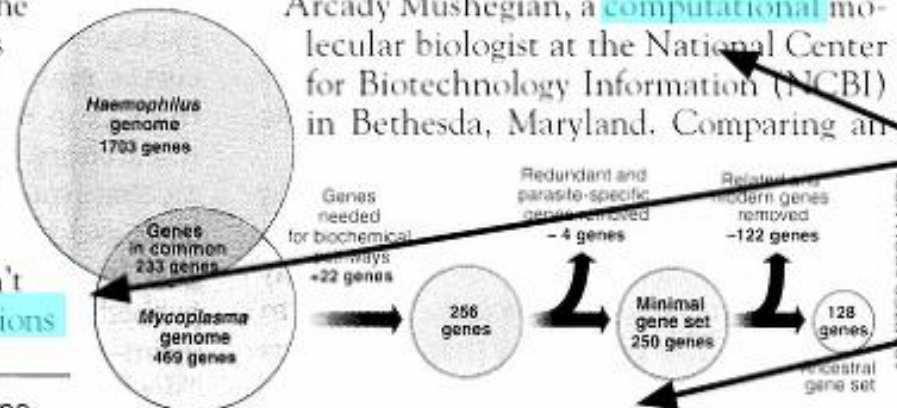
Documents

### Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many **genes** does an **organism** need to **survive**? Last week at the genome meeting here,\* two genome researchers with radically different approaches presented complementary views of the basic genes needed for **life**. One research team, using **computer** analyses to compare known **genomes**, concluded that today's **organisms** can be sustained with just 250 genes, and that the earliest life forms required a mere 128 **genes**. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those **predictions**

"are not all that far apart," especially in comparison to the 75,000 **genes** in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a **genetic numbers** game, particularly as more and more **genomes** are completely mapped and sequenced. "It may be a way of organizing any newly **sequenced genome**," explains Arcady Mushegian, a **computational** molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

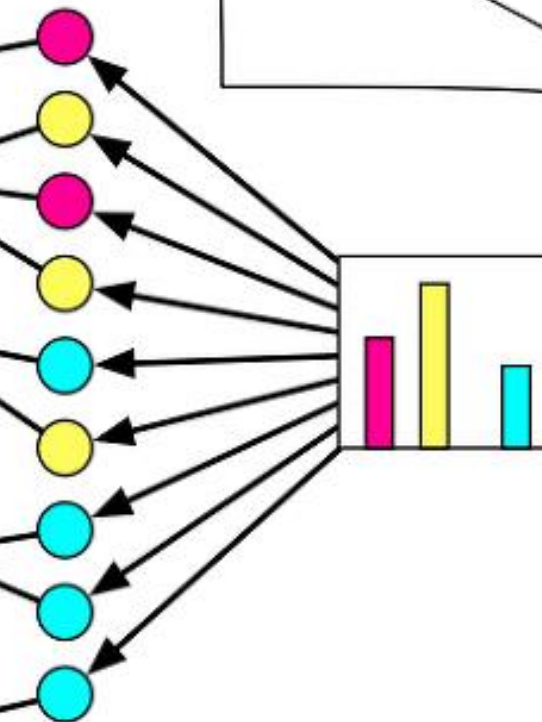


\* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

SCIENCE • VOL. 272 • 24 MAY 1996

Topic proportions & assignments



# #03-1 LDA(Latent Dirichlet Allocation) 개요

## ✓ LDA 예시

문서1 : 저는 사과랑 바나나를 먹어요

문서2 : 우리는 귀여운 강아지가 좋아요

문서3 : 저의 깜찍하고 귀여운 강아지가 바나나를 먹어요



LDA는 각 문서의 토픽 분포와 각 토픽 내의 단어 분포를 추정

### <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%



# #03-2 LDA의 가정

1) 문서에 사용할 단어의 개수  $N$  결정

예) 단어의 개수  $N = 5$

2) 문서에 사용할 토픽의 혼합을 확률 분포에 기반하여 결정

예) 위 예제와 같이 토픽이 2개라고 하였을 때 강아지 토픽을 60%,  
과일 토픽을 40%와 같이 결정

3) 문서에 사용할 각 단어를 (아래와 같이) 정함

3-1) 토픽 분포에서 토픽  $T$ 를 확률적으로 고름

예) 60% 확률로 강아지 토픽을 선택하고, 40% 확률로 과일 토픽을 선택

3-2) 선택한 토픽  $T$ 에서 단어의 출현 확률 분포에 기반해 문서에 사용할 단어를 고름

예) 강아지 토픽을 선택하였다면, 33% 확률로 강아지란 단어를 선택할 수 있음



반복



역공학  
수행

# #03-3 LDA 수행

앞의 문서 생성 과정을 기반으로, LDA는 실제 문서를 보고 문서에 존재하는 단어들로부터 토픽 분포와 단어 분포를 추정

1. 각 문서에 포함된 토픽 비율(문서  $\rightarrow$  토픽 분포)
2. 각 토픽에서 단어가 나올 확률(토픽  $\rightarrow$  단어 분포)

이 과정을 통해 LDA는 주어진 문서 집합에서 잠재된 토픽을 추출

# #03-3 LDA 수행

## 1. 사용자는 토픽의 개수 $k$ 지정

: LDA는 토픽의 개수  $k$ 를 입력받으면,  $k$ 개의 토픽이  $M$ 개의 전체 문서에 걸쳐 분포되어 있다고 가정

## 2. 모든 단어를 $k$ 개 중 하나의 토픽에 할당

: LDA는 모든 문서의 모든 단어에 대해서  $k$ 개 중 하나의 토픽을 랜덤으로 할당

➡ 각 문서는 토픽을 가지며, 토픽은 단어 분포를 가지는 상태

- 랜덤으로 할당하였기 때문에 사실 이 결과는 전부 틀린 상태
- 만약 한 단어가 한 문서에서 2회 이상 등장하였다면, 각 단어는 서로 다른 토픽에 할당되었을 수도 있음



# #03-3 LDA 수행

## 3. 이제 모든 문서의 모든 단어에 대해서 아래의 사항을 반복 진행 (iterative)

- 어떤 문서의 각 단어  $w$ 는 자신은 잘못된 토픽에 할당되어져 있지만, 다른 단어들은 전부 올바른 토픽에 할당되어져 있는 상태라고 가정
- 이에 따라 단어  $w$ 는 아래의 두 가지 기준에 따라서 토픽이 재할당
  1.  $p(\text{topic } t \mid \text{document } d)$  : 문서  $d$ 의 단어들 중 토픽  $t$ 에 해당하는 단어들의 비율
  2.  $p(\text{word } w \mid \text{topic } t)$  : 각 토픽들  $t$ 에서 해당 단어  $w$ 의 분포

➡ 이를 반복하면, 모든 할당이 완료된 수렴 상태

# #03-3 LDA 수행

## 3. 이제 모든 문서의 모든 단어에 대해서 아래의 사항을 반복 진행 (iterative)

- 어떤 문서의 각 단어  $w$ 는 자신은 잘못된 토픽에 할당되어져 있지만, 다른 단어들은 전부 올바른 토픽에 할당되어져 있는 상태라고 가정

- 이에 따라 단어  $w$ 는 아래의 두 가지 기준에 따라서 토픽이 재할당

1.  $p(\text{topic } t \mid \text{document } d)$  : 문서  $d$ 의 단어들 중 토픽  $t$ 에 해당하는 단어들의 비율

2.  $p(\text{word } w \mid \text{topic } t)$  : 각 토픽들  $t$ 에서 해당 단어  $w$ 의 분포

➡ 이를 반복하면, 모든 할당이 완료된 수렴 상태

# #03-3 LDA 수행

## ✓ LDA 토픽 재할당 기준

1.  $p(\text{topic } t \mid \text{document } d)$  : 문서  $d$ 의 단어들 중 토픽  $t$ 에 해당하는 단어들의 비율
2.  $p(\text{word } w \mid \text{topic } t)$  : 각 토픽들  $t$ 에서 해당 단어  $w$ 의 분포

## ✓ 예시

doc1의 세번째 단어 apple의 토픽을 결정하고자 함

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

# #03-3 LDA 수행

## 1. 첫번째 기준

: 문서 doc1의 단어들이 어떤 토픽에 해당하는지를 봄

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

## 2. 두번째 기준

: 단어 apple이 전체 문서에서 어떤 토픽에 할당되어져 있는지를 봄

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

# #03-3 LDA 실습 – 20 newsgroup

## ✓ 실습 개요

20가지 주제 중 오토바이, 야구, 그래픽스, 윈도우, 중동, 기독교, 전자공학, 의학의 8개 주제를 추출하고 이들 텍스트에 LDA 기반의 토픽 모델링 적용

## ✓ 주제 필터링, 벡터화 변환

- LDA 토픽 모델링을 위해 `fetch_20newsgroup( )` API는 `categories` 파라미터를 통해 필요한 주제만 필터링해 추출
- 추출된 텍스트를 Count 기반으로 벡터화 변환  
( LDA는 Count 기반의 벡터화만 사용)

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# 오토바이, 야구, 그래픽스, 윈도우즈, 중동, 기독교, 전자공학, 의학 8개 주제를 추출
cats = ['rec.motorcycles', 'rec.sport.baseball', 'comp.graphics', 'comp.windows.x',
        'talk.politics.mideast', 'soc.religion.christian', 'sci.electronics', 'sci.med']

# 위에서 cats 변수로 기재된 카테고리만 추출
# fetch_20newsgroups( )의 categories에 cats 입력
news_df = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'),
                             categories=cats, random_state=0)

# LDA는 Count 기반의 벡터화만 적용
count_vect = CountVectorizer(max_df=0.95, max_features=1000, min_df=2, stop_words='english', ngram_range=(1,2))
feat_vect = count_vect.fit_transform(news_df.data)
print('CountVectorizer Shape:', feat_vect.shape)

CountVectorizer Shape: (7862, 1000)
```

# #03-3 LDA 실습 – 20 newsgroup

## ✓ LDA 토픽 모델링

```
lda = LatentDirichletAllocation(n_components=8, random_state=0)
lda.fit(feat_vect)
```

▼ LatentDirichletAllocation ⓘ ?  
LatentDirichletAllocation(n\_components=8, random\_state=0)

## ✓ components\_ 속성값

- LatentDirichletAllocation.fit(데이터셋)를 수행  
⇒ LatentDirichletAllocation 객체는 components\_ 속성값을 가지게 됨
- components\_는 개별 토픽별로 각 word 피처가 얼마나 많이 그 토픽에 할당됐는지에 대한 수치를 가지고 있음  
⇒ 높은 값일수록 해당 word 피처는 그 토픽의 중심 word가 됨

# #03-3 LDA 실습 – 20 newsgroup

## ✓ components\_ 형태, 속성 확인

```
print(lda.components_.shape)
lda.components_
(8, 1000)
array([[3.60992018e+01, 1.35626798e+02, 2.15751867e+01, ...,
        3.02911688e+01, 8.66830093e+01, 6.79285199e+01],
       [1.25199920e-01, 1.44401815e+01, 1.25045596e-01, ...,
        1.81506995e+02, 1.25097844e-01, 9.39593286e+01],
       [3.34762663e+02, 1.25176265e-01, 1.46743299e+02, ...,
        1.25105772e-01, 3.63689741e+01, 1.25025218e-01],
       ...,
       [3.60204965e+01, 2.08640688e+01, 4.29606813e+00, ...,
        1.45056650e+01, 8.33854413e+00, 1.55690009e+01],
       [1.25128711e-01, 1.25247756e-01, 1.25005143e-01, ...,
        9.17278769e+01, 1.25177668e-01, 3.74575887e+01],
       [5.49258690e+01, 4.47009532e+00, 9.88524814e+00, ...,
        4.87048440e+01, 1.25034678e-01, 1.25074632e-01]])
```

- 8개의 토픽별로 1000개의 word 피처가 해당 토픽별로 연관도 값을 가지고 있음
- components\_array의 0번째 row, 10번째 col에 있는 값은 Topic #0에 대해서 피처 벡터화된 행렬에서 10번째 칼럼에 해당하는 피처가 Topic #0에 연관되는 수치 값을 가지고 있음



# #03-3 LDA 실습 – 20 newsgroup

## ✓ 토픽별로 연관도가 높은 순으로 Word 나열

- lda\_model.components\_ 값만으로는 각 토픽별 word 연관도를 보기가 어려움
- display\_topics( ) 함수를 만들어 토픽별로 연관도가 높은 순으로 Word 나열

```
def display_topics(model, feature_names, no_top_words):  
    for topic_index, topic in enumerate(model.components_):  
        print('Topic #',topic_index)  
  
        # components_array에서 가장 값이 큰 순으로 정렬했을 때, 그 값의 array 인덱스를 반환  
        topic_word_indexes = topic.argsort()[::-1]  
        top_indexes=topic_word_indexes[:no_top_words]  
  
        # top_indexes대상인 index별로 feature_names에 해당하는 word feature 추출 후 join으로 concat  
        feature_concat = ' '.join([feature_names[i] for i in top_indexes])  
        print(feature_concat)  
  
# CountVectorizer 객체 내의 전체 word의 명칭을 get_features_names( )를 통해 추출  
feature_names = count_vect.get_feature_names_out( )  
  
# 토픽별 가장 연관도가 높은 word를 15개만 추출  
display_topics(lda, feature_names, 15)
```

```
Topic # 0  
year 10 game medical health team 12 20 disease cancer 1993 games years patients good  
Topic # 1  
don just like know people said think time ve didn right going say ll way  
Topic # 2  
image file jpeg program gif images output format files color entry 00 use bit 03  
Topic # 3  
like know don think use does just good time book read information people used post  
Topic # 4  
armenian israel armenians jews turkish people israeli jewish government war dos dos turkey arab armenia 000  
Topic # 5  
edu com available graphics ftp data pub motif mail widget software mit information version sun  
Topic # 6  
god people jesus church believe christ does christian say think christians bible faith sin life  
Topic # 7  
use dos thanks windows using window does display help like problem server need know run
```



# #04 LSA, LDA 비교

특징	LSA	LDA
기법	선형 대수 기반 (SVD)	확률적 생성 모델
출력	잠재 의미 공간의 차원 축소	문서-토픽 분포, 토픽-단어 분포
모델 해석	해석이 어려움	명확한 확률적 해석 가능
노이즈 처리	데이터의 노이즈 감소 효과 있음	토픽 수를 지정하지 않으면 노이즈 포함 가능
새로운 데이터	기존 결과에 업데이트 어려움	쉽게 업데이트 가능
토픽 수 설정	차원 축소 단계에서 암시적으로 설정	사용자 지정 필요
속도	계산 속도가 빠름	느림
적용 사례	정보 검색, 문서 유사도 분석	토픽 모델링, 문서 분류

# 문서 군집화



# #01 문서 군집화 (Document Clustering)

#1 비슷한 텍스트 구성의 문서를 **군집화** (Clustering)

- 군집화 기법(7장)을 텍스트 기반 문서에 적용

#2 텍스트 분류 기반의 문서 분류와 유사

텍스트 분류 기반 문서 분류	문서 군집화
- 학습 데이터로 특정 문서 분류를 학습 -> 모델 생성 (지도 학습)	- 학습 데이터 세트 필요 없음 (비지도 학습)

문서 군집화 수행하기

(Opinion Review 데이터)



# #00 데이터 불러오기

```
[5] import pandas as pd
import glob, os

from google.colab import drive
drive.mount('/content/gdrive/')

path="/content/gdrive/MyDrive/Euron/topics"
# path로 지정한 디렉터리 밑에 있는 모든 .data 파일의 파일명을 리스트로 취합
all_files=glob.glob(os.path.join(path, "*.data"))
filename_list=[]
opinion_text=[]

# 개별 파일의 파일명은 filename_list로 취합
# 개별 파일의 파일 내용은 dataframe 로딩 후 다시 string(문자열)로 변환하여 opinion_text list로 취합
for file_ in all_files:
    # 개별 파일 읽어서 dataframe으로 생성
    df=pd.read_table(file_, index_col=None, header=0, encoding='latin1')

    # 절대 경로로 주어진 파일명을 가공
    filename_ = file_.split('/')[-1] # .split('구분자')
    filename = filename_.split('.')[0] # 맨 마지막 .data 확장자 제거

    # 파일명 list와 파일 내용 list에 파일명과 파일 내용 추가
    filename_list.append(filename) # 파일명 추가
    opinion_text.append(df.to_string()) # 파일 내용 추가

# 파일명 list와 파일 내용 list 객체를 DataFrame으로 생성
document_df = pd.DataFrame({'filename': filename_list, 'opinion_text':opinion_text})
document_df.head()
```

# Opinion Review 데이터  
- 자동차 브랜드, 전자 제품, 호텔 등에 대한 리뷰

	파일명	파일 내용
	filename	opinion_text
0	accuracy_garmin_nuvi_255W_gps	...
1	comfort_honda_accord_2008	...
2	directions_garmin_nuvi_255W_gps	...
3	interior_toyota_camry_2007	...
4	interior_honda_accord_2008	...

# #01 TF-IDF 피처 벡터화

## #1 Lemmatization (어근 변환) 구현 함수: LemNormalize( )

```
from nltk.stem import WordNetLemmatizer
import nltk
import string

remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
lemmar = WordNetLemmatizer()

# 입력으로 들어온 단어들(tokens)에 대해서 lemmatization 어근 변환
def LemTokens(tokens):
    return [lemmar.lemmatize(token) for token in tokens]

# TfidfVectorizer 객체 생성 시 tokenizer 인자로 해당 함수를 설정하여 lemmatization 적용
# 입력으로 문장을 받아서 stop words 제거 -> 소문자 변환 -> 단어 토큰화 -> lemmatization 어근 변환
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

### [텍스트 전처리]

- translate( ): 특정 문자를 지정된 문자로 변환  
\* stop words -> None 변환 (제거 / 정규화)
- lower( ): 소문자 변환 (클렌징)
- word\_tokenize( ) (토큰화)

## #2 피처 벡터화

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english' ,
                             ngram_range=(1,2), min_df=0.05, max_df=0.85 )

#opinion_text 컬럼값으로 피처 벡터화 수행
feature_vect = tfidf_vect.fit_transform(document_df['opinion_text'])
```

- tokenizer: 별도의 함수를 이용하여 토큰화할 때 적용함

- feature\_vect: 개별 문서 텍스트에 대해  
TF-IDF 변환된 피처 벡터화 행렬 도출

# #02 데이터 군집화 (K-Means)

## #1 5개 군집으로 군집화 수행

```
from sklearn.cluster import KMeans

# 5개 집합으로 군집화 수행
km_cluster = KMeans(n_clusters=5, max_iter=10000, random_state=0)
km_cluster.fit(feature_vect)

# 군집의 Label 값
cluster_label = km_cluster.labels_
# 중심별로 할당된 데이터 세트의 좌표 값
cluster_centers = km_cluster.cluster_centers_

# 'cluster_label' 칼럼 추가
document_df['cluster_label'] = cluster_label
document_df.head()
```

	filename	opinion_text	cluster_label
0	accuracy_garmin_nuvi_255W_gps	...	4
1	comfort_honda_accord_2008	...	0
2	directions_garmin_nuvi_255W_gps	...	4
3	interior_toyota_camry_2007	...	0
4	interior_honda_accord_2008	...	0



# #02 데이터 군집화 (K-Means)

# 군집별 결과: 세분화되어 군집화된 경향 있음 (Cluster 2,3)

```
document_df[document_df['cluster_label']==0].sort_values(by='filename')
```

	filename	opinion_text	cluster_label
1	comfort_honda_accord_2008	...	0
16	comfort_toyota_camry_2007	...	0
14	gas_mileage_toyota_camry_2007	...	0
4	interior_honda_accord_2008	...	0
3	interior_toyota_camry_2007	...	0
26	mileage_honda_accord_2008	...	0
27	quality_toyota_camry_2007	...	0
31	seats_honda_accord_2008	...	0
50	sound_ipod_nano_8gb	headphone jack i got a clear case for it an...	0
41	transmission_toyota_camry_2007	...	0

## Cluster 0: 자동차에 대한 리뷰

```
document_df[document_df['cluster_label']==3].sort_values(by='filename')
```

	filename	opinion_text	cluster_label
15	fonts_amazon_kindle	...	3
11	keyboard_netbook_1005ha	...	3
46	size_asus_netbook_1005ha	...	3

## Cluster 3: 킨들, 넷북에 대한 리뷰 포함

```
document_df[document_df['cluster_label']==1].sort_values(by='filename')
```

	filename	opinion_text	cluster_label
5	bathroom_bestwestern_hotel_sfo	...	1
8	food_holiday_inn_london	...	1
12	food_swissotel_chicago	...	1
6	free_bestwestern_hotel_sfo	...	1
24	location_bestwestern_hotel_sfo	...	1
29	location_holiday_inn_london	...	1
40	parking_bestwestern_hotel_sfo	...	1
25	price_amazon_kindle	...	1
39	price_holiday_inn_london	...	1
35	room_holiday_inn_london	...	1
37	rooms_bestwestern_hotel_sfo	...	1
23	rooms_swissotel_chicago	...	1
34	service_bestwestern_hotel_sfo	...	1
22	service_holiday_inn_london	...	1
36	service_swissotel_hotel_chicago	...	1
42	staff_bestwestern_hotel_sfo	...	1
44	staff_swissotel_chicago	...	1

## Cluster 1: 호텔에 대한 리뷰

## Cluster 2: 휴대용 전자기기에 대한 리뷰

```
document_df[document_df['cluster_label']==2].sort_values(by='filename')
```

	filename	opinion_text	cluster_label
7	battery-life_amazon_kindle	...	2
17	battery-life_ipod_nano_8gb	...	2
10	battery-life_netbook_1005ha	...	2
18	features_windows7	...	2
32	performance_honda_accord_2008	...	2
20	performance_netbook_1005ha	...	2
48	speed_windows7	...	2
45	video_ipod_nano_8gb	...	2

## Cluster 4: 차량용 네비게이션에 대한 리뷰

```
document_df[document_df['cluster_label']==4].sort_values(by='filename')
```

	filename	opinion_text	cluster_label
0	accuracy_garmin_nuvi_255W_gps	...	4
19	buttons_amazon_kindle	...	4
2	directions_garmin_nuvi_255W_gps	...	4
13	display_garmin_nuvi_255W_gps	...	4
9	eyesight-issues_amazon_kindle	...	4
38	navigation_amazon_kindle	...	4
21	satellite_garmin_nuvi_255W_gps	...	4
28	screen_garmin_nuvi_255W_gps	...	4
30	screen_ipod_nano_8gb	...	4
33	screen_netbook_1005ha	...	4
43	speed_garmin_nuvi_255W_gps	...	4
49	updates_garmin_nuvi_255W_gps	...	4
47	voice_garmin_nuvi_255W_gps	...	4



# #02 데이터 군집화 (K-Means)

## #2 3개 군집으로 군집화 수행

```
from sklearn.cluster import KMeans

# 3개의 집합으로 군집화
km_cluster = KMeans(n_clusters=3, max_iter=10000, random_state=0)
km_cluster.fit(feature_vect)
cluster_label = km_cluster.labels_
cluster_centers = km_cluster.cluster_centers_

# 소속 클러스터를 cluster_label 컬럼으로 할당하고 cluster_label 값으로 정렬
document_df['cluster_label'] = cluster_label
document_df.sort_values(by='cluster_label')
```

# #02 데이터 군집화 (K-Means)

# 군집별 결과: 대체로 잘 군집화됨

## Cluster 0: 자동차에 대한 리뷰

	filename	opinion_text	cluster_label
50	sound_ipod_nano_8gb	headphone jack i got a clear case for it an...	0
27	quality_toyota_camry_2007	...	0
26	mileage_honda_accord_2008	...	0
41	transmission_toyota_camry_2007	...	0
31	seats_honda_accord_2008	...	0
14	gas_mileage_toyota_camry_2007	...	0
16	comfort_toyota_camry_2007	...	0
32	performance_honda_accord_2008	...	0
1	comfort_honda_accord_2008	...	0
4	interior_honda_accord_2008	...	0
3	interior_toyota_camry_2007	...	0

## Cluster 1: 호텔에 대한 리뷰

34	service_bestwestern_hotel_sfo	...	1
29	location_holiday_inn_london	...	1
35	room_holiday_inn_london	...	1
36	service_swissotel_hotel_chicago	...	1
24	location_bestwestern_hotel_sfo	...	1
23	rooms_swissotel_chicago	...	1
22	service_holiday_inn_london	...	1
37	rooms_bestwestern_hotel_sfo	...	1
39	price_holiday_inn_london	...	1
40	parking_bestwestern_hotel_sfo	...	1
42	staff_bestwestern_hotel_sfo	...	1
5	bathroom_bestwestern_hotel_sfo	...	1
12	food_swissotel_chicago	...	1
44	staff_swissotel_chicago	...	1
8	food_holiday_inn_london	...	1
6	free_bestwestern_hotel_sfo	...	1

## Cluster 2: 휴대용 전자기기에 대한 리뷰

38	navigation_amazon_kindle	...	2
43	speed_garmin_nuvi_255W_gps	...	2
48	speed_windows7	...	2
45	video_ipod_nano_8gb	...	2
46	size_asus_netbook_1005ha	...	2
47	voice_garmin_nuvi_255W_gps	...	2
0	accuracy_garmin_nuvi_255W_gps	...	2
30	screen_ipod_nano_8gb	...	2
2	directions_garmin_nuvi_255W_gps	...	2
7	battery-life_amazon_kindle	...	2
9	eyesight-issues_amazon_kindle	...	2
10	battery-life_netbook_1005ha	...	2
11	keyboard_netbook_1005ha	...	2
13	display_garmin_nuvi_255W_gps	...	2
33	screen_netbook_1005ha	...	2
15	fonts_amazon_kindle	...	2
18	features_windows7	...	2
19	buttons_amazon_kindle	...	2
20	performance_netbook_1005ha	...	2
21	satellite_garmin_nuvi_255W_gps	...	2
49	updates_garmin_nuvi_255W_gps	...	2
28	screen_garmin_nuvi_255W_gps	...	2

# 군집별 핵심단어 추출



# #01 군집의 핵심 단어 확인

## #1 군집 내 단어와 중심의 거리 확인

```
cluster_centers = km_cluster.cluster_centers_  
print('cluster_centers shape :', cluster_centers.shape)  
print(cluster_centers)
```

```
cluster_centers shape : (3, 4610)  
[[0.      0.00084138 0.      ... 0.      0.      0.      ]  
 [0.      0.00099548 0.00174656 ... 0.      0.00183397 0.00144581]  
 [0.01047343 0.      0.      ... 0.00735716 0.      0.      ]]
```

- cluster\_centers\_: 군집을 구성하는 단어가 중심과 얼마나 가까운지

\* 배열 (행: 개별 군집, 열: 개별 피처): 1에 가까울 수록 중심과 가까움

- 군집: 3개, word 피처: 4610개

# #01 군집의 핵심 단어 확인

## #2 필요한 함수 정의

```
# 군집별 top n 핵심단어, 그 단어의 중심 위치 상대값, 대상 파일명들을 반환함.
def get_cluster_details(cluster_model, cluster_data, feature_names, clusters_num, top_n_features=10):
    cluster_details = {}

    # cluster_centers 배열 내 값이 큰 순으로 정렬된 index 값을 반환
    # 군집 중심점(centroid)별 할당된 word 피쳐들의 거리값이 큰 순으로 값을 구하기 위함.
    centroid_feature_ordered_ind = cluster_model.cluster_centers_.argsort()[::-1]

    # 개별 군집별로 반복하면서 핵심단어, 그 단어의 중심 위치 상대값, 대상 파일명 입력
    for cluster_num in range(clusters_num):
        # 개별 군집별 정보를 담을 데이터 초기화.
        cluster_details[cluster_num] = {}
        cluster_details[cluster_num]['cluster'] = cluster_num

        # cluster_centers_.argsort()[::-1] 로 구한 index 를 이용하여 top n 피쳐 단어를 구함.
        top_feature_indexes = centroid_feature_ordered_ind[cluster_num, :top_n_features]
        top_features = [ feature_names[ind] for ind in top_feature_indexes ]

        # top_feature_indexes를 이용해 해당 피쳐 단어의 중심 위치 상대값 구함
        top_feature_values = cluster_model.cluster_centers_[cluster_num, top_feature_indexes].tolist()

        # cluster_details 딕셔너리 객체에 개별 군집별 핵심 단어와 중심위치 상대값, 그리고 해당 파일명 입력
        cluster_details[cluster_num]['top_features'] = top_features
        cluster_details[cluster_num]['top_features_value'] = top_feature_values
        filenames = cluster_data[cluster_data['cluster_label'] == cluster_num]['filename']
        filenames = filenames.values.tolist()

        cluster_details[cluster_num]['filenames'] = filenames

    return cluster_details
```

- get\_cluster\_details( )  
: 군집 정보(번호, 핵심단어, 중심위치 상대값, 파일명 ...)  
딕셔너리 형태로 반환

```
def print_cluster_details(cluster_details):
    for cluster_num, cluster_detail in cluster_details.items():
        print('##### Cluster {0}'.format(cluster_num)) # 군집번호
        print('Top features:', cluster_detail['top_features']) # 핵심단어
        print('Reviews 파일명 :', cluster_detail['filenames'][:7]) # 파일명
        print('=====')
```

- print\_cluster\_details( ): 군집 정보를 보기 좋게 표현

# #01 군집의 핵심 단어 확인

## #3 결과 확인

```
# 피처명 리스트
feature_names = tfidf_vect.get_feature_names_out()

# 군집 정보
cluster_details = get_cluster_details(cluster_model=km_cluster, cluster_data=document_df,\
                                     feature_names=feature_names, clusters_num=3, top_n_features=10 )

print_cluster_details(cluster_details)
```

# 군집 3개 군집화 결과 이용

```
##### Cluster 0 Cluster 0: 자동차에 대한 리뷰
p features: ['interior', 'seat', 'mileage', 'comfortable', 'gas', 'quality', 'gas mileage', 'transmission', 'car', 'performance']
views 파일명 : ['comfort_honda_accord_2008', 'interior_toyota_camry_2007', 'interior_honda_accord_2008', 'gas_mileage_toyota_camry_2007', 'comfort_toyota_camry_2007', 'mileage_honda_accord_2008',
=====
##### Cluster 1 Cluster 1: 호텔에 대한 리뷰
p features: ['room', 'hotel', 'service', 'staff', 'food', 'location', 'bathroom', 'clean', 'price', 'parking']
views 파일명 : ['bathroom_bestwestern_hotel_sfo', 'free_bestwestern_hotel_sfo', 'food_holiday_inn_london', 'food_swissotel_chicago', 'service_holiday_inn_london', 'rooms_swissotel_chicago', 'location_bestwe:
=====
##### Cluster 2 Cluster 2: 휴대용 전자기기에 대한 리뷰
p features: ['screen', 'battery', 'keyboard', 'battery life', 'life', 'kindle', 'direction', 'voice', 'size', 'map']
views 파일명 : ['accuracy_garmin_nuvi_255W_gps', 'directions_garmin_nuvi_255W_gps', 'battery-life_amazon_kindle', 'eyesight-issues_amazon_kindle', 'battery-life_netbook_1005ha', 'keyboard_netbook_1005h
=====
```

### [핵심 단어]

Cluster 0: 'interior', 'seat', 'size', 'mileage', 'comfortable'  
Cluster 1: 'room', 'hotel', 'service', 'staff', 'clean', 'price'  
Cluster 2: 'screen', 'battery life'

# 문서 유사도

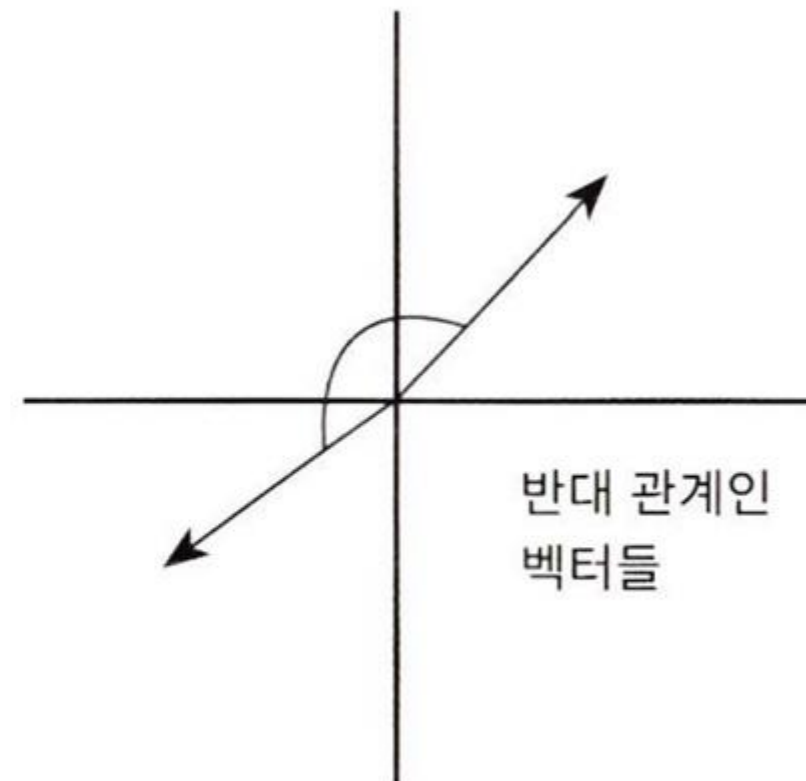
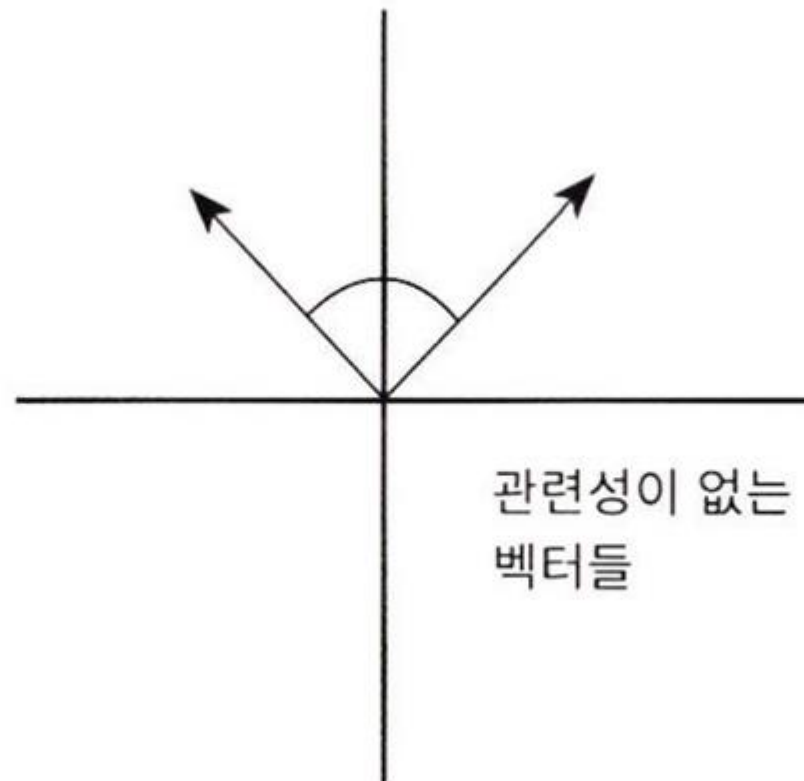
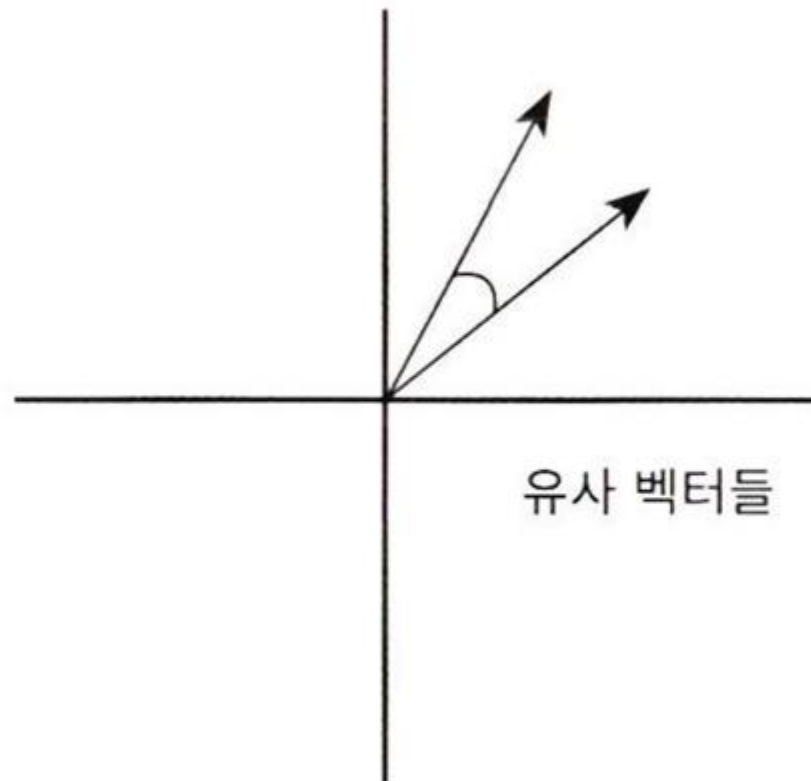




# #코사인 유사도

**코사인 유사도:** 두 벡터 사이의 사잇각을 구해서 얼마나 유사한지 수치로 적용한 것  
- 벡터의 크기보다는 벡터의 상호 방향성이 얼마나 유사한지에 기반

두 벡터의 사잇각





# #코사인 유사도

코사인 유사도를 구하는 방법

$$A * B = \|A\| \|B\| \cos \theta$$



$$\text{similarity} = \cos \theta = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

문서의 유사도 비교에 코사인 유사도를 사용하는 이유

- 문서를 피처 벡터화 변환하면 차원이 매우 많은 희소 행렬이 되기 쉬움
- 문서가 매우 긴 경우 단어의 빈도수도 더 많을 것, 빈도수에만 기반해서는 공정한 비교 X

# #코사인 유사도

## 코사인 유사도 기반으로 문서 유사도 구해보기

```
import numpy as np

def cos_similarity(v1,v2):
    dot_product=np.dot(v1,v2)
    l2_norm=(np.sqrt(sum(np.square(v1)))*np.sqrt(sum(np.square(v2))))
    similarity=dot_product/l2_norm

    return similarity
```

cos\_similarity() 함수: 두 개의 넘파이 배열에 대한 코사인 유사도를 구함

```
from sklearn.feature_extraction.text import TfidfVectorizer

doc_list=['if you take the blue pill, the story ends',
          'if you take the red pill, you stay in Wonderland',
          'if you take the red pill, I show you how deep the rabbit hole goes']

tfidf_vect_simple=TfidfVectorizer()
feature_vect_simple=tfidf_vect_simple.fit_transform(doc_list)
print(feature_vect_simple.shape)
```

문서를 TF-IDF로 벡터화된 행렬로 변환

(3, 18)

# #코사인 유사도

```
#TFidfVectorizer로 transform()한 결과는 희소 행렬이므로 밀집 행렬로 변환.  
feature_vect_dense=feature_vect_simple.todense()
```

```
#첫 번째 문장과 두 번째 문장의 피쳐 벡터 추출  
vect1=np.array(feature_vect_dense[0]).reshape(-1,)  
vect2=np.array(feature_vect_dense[1]).reshape(-1,)
```

```
#첫 번째 문장과 두 번째 문장의 피쳐 벡터로 두 개 문장의 코사인 유사도 추출  
similarity_simple=cos_similarity(vect1, vect2)  
print('문장 1, 문장 2 Cosine 유사도: {0:.3f}'.format(similarity_simple))
```

문장 1, 문장 2 Cosine 유사도: 0.402

```
vect1=np.array(feature_vect_dense[0]).reshape(-1,)  
vect3=np.array(feature_vect_dense[2]).reshape(-1,)  
similarity_simple=cos_similarity(vect1, vect3)  
print('문장 1, 문장 3 Cosine 유사도: {0:.3f}'.format(similarity_simple))
```

문장 1, 문장 3 Cosine 유사도: 0.404  
문장 2, 문장 3 Cosine 유사도: 0.456

```
vect2=np.array(feature_vect_dense[1]).reshape(-1,)  
vect3=np.array(feature_vect_dense[2]).reshape(-1,)  
similarity_simple=cos_similarity(vect2, vect3)  
print('문장 2, 문장 3 Cosine 유사도: {0:.3f}'.format(similarity_simple))
```

# #코사인 유사도

```
from sklearn.metrics.pairwise import cosine_similarity

similarity_simple_pair=cosine_similarity(feature_vect_simple[0], feature_vect_simple)
print(similarity_simple_pair)
```

```
[[1.          0.40207758 0.40425045]]
```

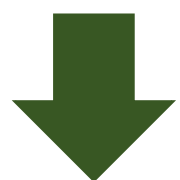


1이라는 값을 제거

```
from sklearn.metrics.pairwise import cosine_similarity

similarity_simple_pair=cosine_similarity(feature_vect_simple[0], feature_vect_simple[1:])
print(similarity_simple_pair)
```

```
[[0.40207758 0.40425045]]
```



쌍으로 코사인 유사도 값 제공

```
similarity_simple_pair=cosine_similarity(feature_vect_simple, feature_vect_simple)
print(similarity_simple_pair)
print('shape:', similarity_simple_pair.shape)
```

```
[[1.          0.40207758 0.40425045]
 [0.40207758 1.          0.45647296]
 [0.40425045 0.45647296 1.          ]]
shape: (3, 3)
```

# #코사인 유사도

## Opinion Review 데이터 세트를 이용한 문서 유사도 측정

```
import pandas as pd
import glob, os
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

path=r'/content/drive/MyDrive/topics'
all_files=glob.glob(os.path.join(path, "*.data"))
filename_list=[]
opinion_text=[]

for file_ in all_files:
    df=pd.read_table(file_,index_col=None, header=0, encoding='latin1')
    filename_=file_.split('###')[-1]
    filename=filename_.split('.')[0]
    filename_list.append(filename)
    opinion_text.append(df.to_string())

document_df=pd.DataFrame({'filename':filename_list, 'opinion_text':opinion_text})

tfidf_vect=TfidfVectorizer(tokenizer=LemNormalize, stop_words='english',#
                             ngram_range=(1,2), min_df=0.05, max_df=0.85)
feature_vect=tfidf_vect.fit_transform(document_df['opinion_text'])

km_cluster=KMeans(n_clusters=3, max_iter=10000, random_state=0)
km_cluster.fit(feature_vect)
cluster_label=km_cluster.labels_
cluster_centers=km_cluster.cluster_centers_
document_df['cluster_label']=cluster_label
```

데이터 세트를 DataFrame으로 로드,  
문서 군집화 적용

# #코사인 유사도

```
from sklearn.metrics.pairwise import cosine_similarity

#cluster_label=1인 데이터는 호텔로 군집화된 데이터임. DataFrame에서 해당 인덱스를 추출
hotel_indexes=document_df[document_df['cluster_label']==1].index
print('호텔로 군집화 된 문서들의 DataFrame Index:', hotel_indexes)

#호텔로 군집화된 데이터 중 첫 번째 문서를 추출해 파일명 표시.
comparison_docname=document_df.iloc[hotel_indexes[0]]['filename']
print('##### 비교 기준 문서명', comparison_docname, '와 타 문서 유사도#####')

'''document_df에서 추출한 Index 객체를 feature_vect로 입력해 호텔 군집화된 feature_vect 추출
이를 이용해 호텔로 군집화된 문서 중 첫 번째 문서와 다른 문서간의 코사인 유사도 측정.'''
similarity_pair=cosine_similarity(feature_vect[hotel_indexes[0]], feature_vect[hotel_indexes])
print(similarity_pair)
```

: document\_df에서 호텔로 군집화된 문서의 인덱스를 추출

: 추출된 인덱스를 이용해 feature\_vect에서 호텔로 군집화된 문서의 피쳐 벡터를 추출

```
호텔로 군집화 된 문서들의 DataFrame Index: Index([2, 4, 6, 10, 17, 19, 21, 22, 26, 28, 31, 35, 38, 39, 41, 44], dtype='int64')
##### 비교 기준 문서명 performance_netbook_1005ha 와 타 문서 유사도#####
[[1.          0.02686658 0.03396213 0.10132148 0.02730274 0.02828863
  0.0153132   0.37522154 0.29839752 0.31205868 0.06579626 0.13663351
  0.05319598 0.20211304 0.14230545 0.06275963]]
```

# #코사인 유사도

첫 번째 문서와 다른 문서 간에 유사도가 높은 순으로 정렬 및 시각화

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

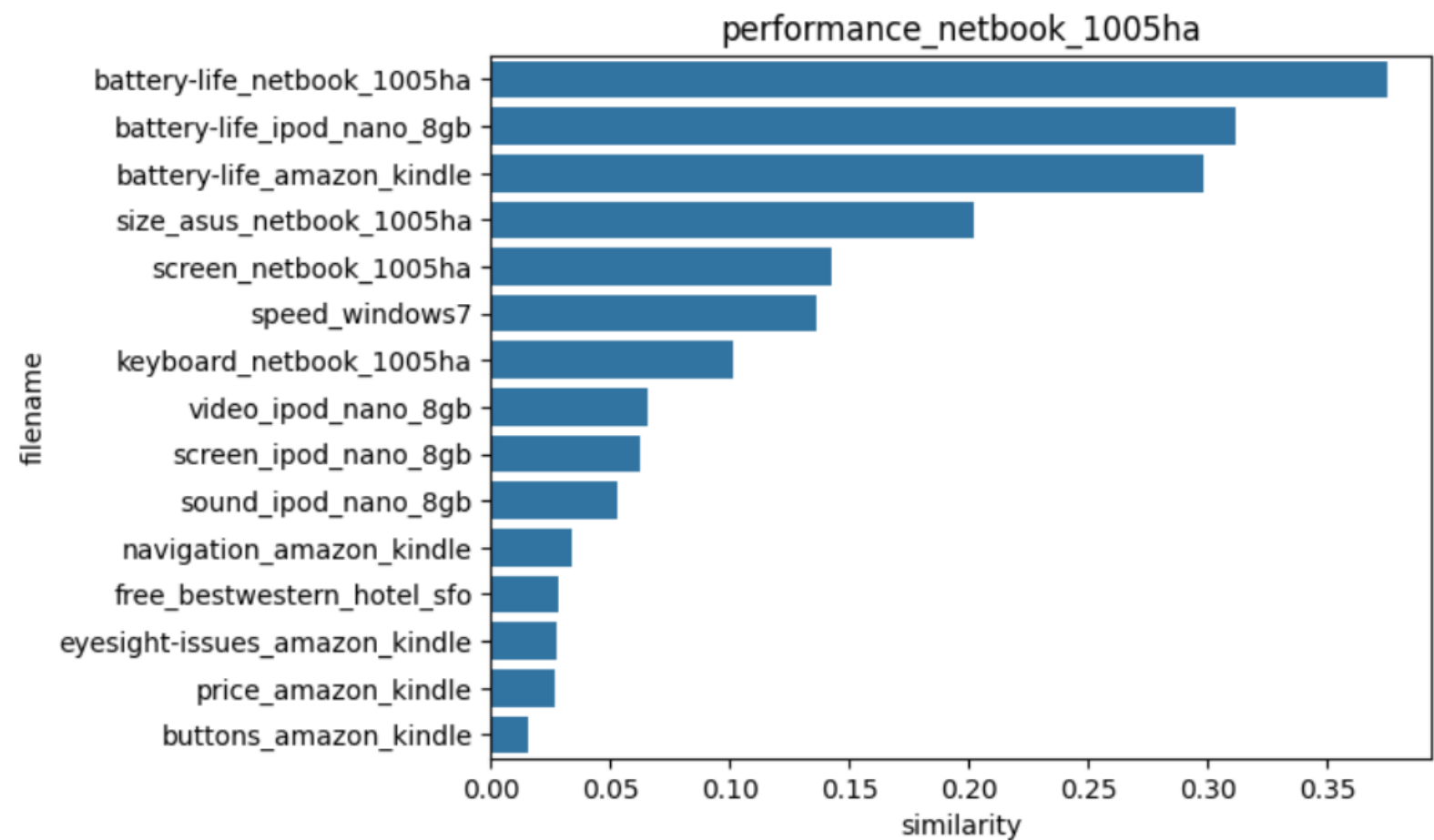
#첫 번째 문서와 타 문서 간 유사도가 큰 순으로 정렬한 인덱스를 추출하되 자기 자신은 제외
sorted_index=similarity_pair.argsort()[::-1]
sorted_index=sorted_index[:, 1:]
print(sorted_index)

#유사도가 큰 순으로 hotel_indexes를 추출하여 재 정렬.
print(hotel_indexes)
hotel_sorted_indexes = hotel_indexes[sorted_index.reshape(-1,)]

# 유사도가 큰 순으로 유사도 값을 재정렬하되 자기 자신은 제외
hotel_1_sim_value = np.sort(similarity_pair.reshape(-1,))[::-1]
hotel_1_sim_value = hotel_1_sim_value[1:]

# 유사도가 큰 순으로 정렬된 Index와 유사도값을 이용하여 파일명과 유사도값을 Seaborn 막대 그래프로 시각화
hotel_1_sim_df = pd.DataFrame()
hotel_1_sim_df['filename'] = document_df.iloc[hotel_sorted_indexes]['filename']
hotel_1_sim_df['similarity'] = hotel_1_sim_value

sns.barplot(x='similarity', y='filename',data=hotel_1_sim_df)
plt.title(comparison_docname)
```



# 한글 텍스트 처리





# #한글 텍스트 처리

## 한글 NLP 처리의 어려움

일반적으로 한글 언어 처리는 라틴어 처리보다 어려움 - 띄어쓰기와 다양한 조사 때문

-> 띄어쓰기: 한글은 띄어쓰기를 잘못하면 의미가 왜곡되어 전달될 수 있음

-> 조사: 어근 추출 등의 전처리 시 제거하기가 까다로움

## KoNLPy 소개

**KoNLPy**: 파이썬의 대표적인 한글 형태소 패키지

-> 기존의 C/C++, Java로 만들어진 한글 형태소 엔진을 파이썬 래퍼 기반으로 재작성

# #한글 텍스트 처리

## KoNLPy 설치

### 윈도우

1. Java 1.7+이 설치되어 있나요?
2. JAVA\_HOME 설정하기
3. JType1 (>=0.5.7)을 다운로드 받고 설치. 다운 받은 *.whl* 파일을 설치하기 위해서는 *pip* 을 업그레이드 해야할 수 있습니다.

```
> pip install --upgrade pip
> pip install JType1-0.5.7-cp27-none-win_amd64.whl
```

### <Jtype1 모듈 설치>

#### 1) 아나콘다 이용

```
conda install -c conda-forge jtype1
```

#### 2) pip 이용

```
pip install --upgrade pip
```

```
pip install JType1-0.6.3-cp36-cp36m-win_amd64.whl
```

# #한글 텍스트 처리

## <Java 환경 설정>

Java 설치 -> JAVA\_HOME 설정하기

- 일반적으로 압축이 풀리는 기준 JDK 폴더를 JAVA\_HOME으로 설정
- KoNLPy의 경우: jvm.dll이 들어 있는 폴더를 설정

## <KoNLPy 설치>

Pip install konlpy

# #한글 텍스트 처리

## 네이버 영화 평점 감성 분석

```
import pandas as pd

train_df=pd.read_csv('ratings_train.txt', sep='\\t')
train_df.head(3)
```

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	0

ratings\_train.txt 파일을 DataFrame으로 로딩

```
train_df['label'].value_counts( )
```

	count
label	
0	75173
1	74827

0과 1의 Label 값 비율: 1이 긍정, 0이 부정

# #한글 텍스트 처리

```
import re

train_df=train_df.fillna(' ')
#정규 표현식을 이용해 숫자를 공백으로 변경(정규 표현식으로 #d 는 숫자를 의미함.)
train_df['document']=train_df['document'].apply(lambda x:re.sub(r"#d+", " ", x))

#테스트 데이터 셋을 로딩하고 동일하게 Null 및 숫자를 공백으로 변환
test_df=pd.read_csv('ratings_test.txt', sep='#t')
test_df=test_df.fillna(' ')
test_df['document']=test_df['document'].apply(lambda x:re.sub(r"#d+", " ", x))

#id 칼럼 삭제 수행
train_df.drop('id', axis=1, inplace=True)
test_df.drop('id', axis=1, inplace=True)
```

Null 값을 공백으로 변환, 숫자의 경우 정규 표현식 모듈 re를 이용해 공백으로 변환

```
from konlpy.tag import Twitter

twitter=Twitter()
def tw_tokenizer(text):
    #입력 인자로 들어온 텍스트를 형태소 단어로 토큰화해 리스트 형태로 반환
    tokens_ko=twitter.morphs(text)
    return tokens_ko
```

morphs() 메서드: 입력 인자로 들어온 문장을 형태소 단어 형태로 토큰화해 list 객체로 반환

# #한글 텍스트 처리

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

#Twitter 객체의 morphs( ) 객체를 이용한 tokenizer를 사용. ngram_range는 (1,2)
tfidf_vect=TfidfVectorizer(tokenizer=tw_tokenizer, ngram_range=(1,2), min_df=3, max_df=0.9)
tfidf_vect.fit(train_df['document'])
tfidf_matrix_train=tfidf_vect.transform(train_df['document'])
```

TfidfVectorizer를 이용해  
TF-IDF 피쳐 모델 생성

```
#로지스틱 회귀를 이용해 감성 분석 분류 수행.
lg_clf=LogisticRegression(random_state=0)

#파라미터 C 최적화를 위해 GridSearchCV를 이용.
params={'C': [1, 3.5, 4.5, 5.5, 10]}
grid_cv=GridSearchCV(lg_clf, param_grid=params, cv=3, scoring='accuracy', verbose=1)
grid_cv.fit(tfidf_matrix_train, train_df['label'])
print(grid_cv.best_params_, round(grid_cv.best_score_, 4))
```

로지스틱 회귀를 이용해  
분류 기반의 감성 분석 수행

```
{ 'C': 3.5} 0.8596
```

# #한글 텍스트 처리

테스트 데이터 세트를 이용한 최종 감성 분석 예측 수행

```
from sklearn.metrics import accuracy_score

#학습 데이터를 적용한 TfidfVectorizer를 이용해 테스트 데이터를 TF-IDF 값으로 피쳐 변환함.
tfidf_matrix_test=tfidf_vect.transform(test_df['document'])

#classifier는 GridSearchCV에서 최적 파라미터로 학습된 classifier를 그대로 이용
best_estimator=grid_cv.best_estimator_
preds=best_estimator.predict(tfidf_matrix_test)

print('Logistic Regression 정확도: ',accuracy_score(test_df['label'],preds))
```

Logistic Regression 정확도: 0.86182

# THANK YOU

