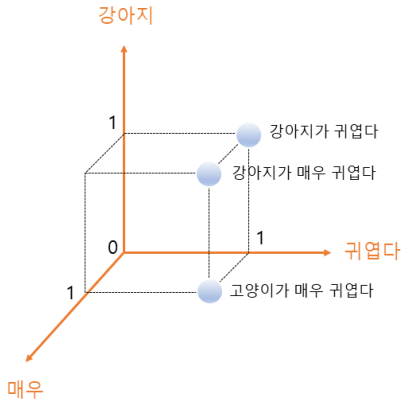
# 4. 주제 분석

## 4.1. 문서 유사도

아래과 같은 TDM이 있다고 해보자.

| | 강아지 | 귀엽다 | 매우 |
|---|---|---|---|
| 강아지가 귀엽다 | 1 | 1 | 0 |
| 강아지가 매우 귀엽다 | 1 | 1 | 1 |
| 고양이가 매우 귀엽다 | 0 | 1 | 1 |

"강아지가 귀엽다"는 (1, 1, 0), "강아지가 매우 귀엽다"는 (1, 1, 1), "고양이가 매우 귀엽다"는 (0, 1, 1)의 좌표를 가지는 것으로 볼 수 있다. 즉, 각 단어를 축으로 하는 특성 공간(feature space)에서 문서들을 하나의 위치로 생각할 수 있다.




그렇다면 특성 공간 상에서 거리를 이용해 두 문서의 유사성(similarity)을 측정할 수 있다. 유사도를 계산하는 방법에는 여러 가지가 있다.

- 유클리드 거리(euclidean distance)
- 코사인 유사도(cosine similarity)
- 맨해튼 거리(Manhattan distance)
- 자카드 유사도(Jaccard similarity)

여기서는 유클리드 거리와 코사인 유사도에 대해 알아보자.

### 4.1.1. 유클리드 거리

유클리드 거리는 다음과 같은 방식으로 계산한다.

$$\text{euclidean\_distance}(a, b) = \sqrt{\sum_i (a_i - b_i)^2}$$

문서의 좌표는 각 단어의 출현 빈도이므로, 모든 단어의 출현 빈도가 같다면 유클리드 거리는 0이 된다. 유클리드 거리는 단어의 빈도가 늘어나거나 줄어들면, 의미에 차이가 없어도 거리가 멀어진다. 예를 들어 "강아지 귀여워. 강아지 귀여워."와 "고양이 귀여워"는 "강아지 귀여워."와 똑같은 거리에 있게 된다. 왜냐하면 '강아지'와 '귀여워'가 1씩 증가한 것이나, '강아지'가 1 감소하고 '고양이'가 1증가한 것이나 거리상으로는 동일하기 때문이다.

### 4.1.2. 코사인 유사도

코사인 유사도는 원점(모든 단어의 빈도가 0인 경우)에서 보았을 때 두 문서의 각도에 바탕을 둔 거리 측정 방식이다. 만약 두 문서의 각도가 $\theta$라면 $\cos\theta$를 거리로 삼는 방식이다. $0^\circ$면 $\cos 0^\circ = 1$로 최대가 되고, $180^\circ$면 $\cos 180^\circ = -1$로 최소가 된다. 거리와 반대로 비슷할 수록 값이 커진다.

$$\text{cosine\_similarity}(a, b) = \frac{a \cdot b}{\|a\|\|b\|} = \frac{\sum_i a_i \times b_i}{\sqrt{\sum_i a_i^2} \times \sqrt{\sum_i b_i^2}}$$

앞서 "강아지 귀여워"와 "강아지 귀여워, 강아지 귀여워."의 경우 단지 모든 단어가 2배로 늘어났을 뿐이기 때문에 두 문서의 좌표는 원점에서 봤을 때 방향이 같다. 따라서 유사도가 1이 된다.

### 4.1.3. 유사도 계산 실습

```
from sklearn.externals import joblib
import pandas
```

```
def get_nouns():
    pass
```

```
with open('ai_news.pkl', 'rb') as f:
    data = joblib.load(f)
locals().update(data)
```

```
article = pandas.read_csv('인공지능 관련 뉴스메타데이터(2013.01.01-2017.08.31).CSV',
                          encoding='cp949', engine='python')
```

#### 유클리드 거리

```
from sklearn.metrics.pairwise import euclidean_distances
```

0번 기사와 1번 기사의 거리를 구한다.

```
euclidean_distances(tdm[0], tdm[1])
```

```
array([[6.244998]])
```

0번 문서와 유클리드 거리가 가장 가까운 문서를 찾는다.

```
import numpy as np
```

```
dist = euclidean_distances(tdm[0], tdm[1:])
np.argmin(dist) + 1
```

```
357
```

```
article.loc[0, '본문']
```

```
'지난해 모바일 게임 열풍에 잠시 주춤했던 온라인 게임이 새해를 맞아 화려한 반격을 준비하고 있다. 새해 벽두 ‘아키에이지’를 시
```

```
article.loc[357, '본문']
```

```
'FPS(슈팅게임) 시장의 새 판 짜기가 가능할까  '서든어택'과 '스페셜포스'가 여전히 난공불락의 진지를 지키고 있는 FPS장르에서 ㅂ
```

#### 코사인 유사도

```
from sklearn.metrics.pairwise import cosine_similarity
```

이번에는 0번 기사와 1번 기사의 코사인 유사도를 구해본다.

```
cosine_similarity(tdm[0], tdm[1])
```

```
array([[0.048795]])
```

0번 기사와 유사도 가장 높은 문서를 찾는다.

```
sim = cosine_similarity(tdm[0], tdm[1:])
```

```
np.argmax(sim) + 1
```

```
1394
```

```
article.loc[1394, '본문']
```

```
'[게임] 축구게임으로 브라질월드컵 열기 즐겨라!  "대~한민국!"을 외치는 함성 소리가 가득 찰 브라질월드컵을 앞두고, 게임업체들(
```

## 4.2. 문서 클러스터링

```
import pandas as pd
from sklearn.externals import joblib
```

```
with open('amazon.pkl', 'rb') as f:
    data = joblib.load(f)
locals().update(data)
```

```
df = pd.read_csv('amazon_cells_labelled.txt', sep="\t", header=None)
```

### 4.2.1. 스펙트럴 클러스터링

```
from sklearn.cluster import SpectralClustering
```

문서들을 유클리드 거리를 기준으로 4개의 클러스터( `n_clusters` )로 묶는다.

```
cl = SpectralClustering(n_clusters=4, random_state=1234)
```

```
labels = cl.fit_predict(tdm[:100])
```

```
labels
```

```
array([0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 1, 3, 1, 3, 0, 0,
       0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 3, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 3, 0, 1, 3, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       3, 0, 0, 0, 1, 0, 0, 0, 0, 3, 0, 2])
```

```
from operator import itemgetter
```

각 클러스터별로 많이 나온 단어들을 확인한다.

```
words = vectorizer.get_feature_names()

def top10(labels):
    freq_words = []
    for i in range(4):
        count = tdm[labels == i, :].sum(axis=0)
        ws = [w for w, n in sorted(zip(words, count.flat), key=itemgetter(1), reverse=True)[:10]]
        freq_words.append(ws)

    return pd.DataFrame(freq_words)
```

```
top10(labels)
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | phone | battery | good | ve | time | right | money | doesn | use | headsets |
| **1** | great | works | worked | item | phone | choice | jawbone | mic | quality | situations |
| **2** | disappointed | decision | battery | 10 | 100 | 11 | 12 | 13 | 15 | 15g |
| **3** | product | love | worthless | impressed | thing | waaay | sensitive | big | handy | cheaper |

코사인 유사도를 바탕으로 클러스터링해보자.

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
cl_cos = SpectralClustering(n_clusters=4, affinity=cosine_similarity, random_state=1234)
```

```
labels_cos = cl_cos.fit_predict(tdm[:100])
```

```
top10(labels_cos)
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | phone | good | doesn | use | headsets | case | sturdy | ve | way | hear |
| **1** | great | works | worked | item | phone | choice | jawbone | mic | quality | situation |
| **2** | battery | disappointed | decision | runs | right | problem | bought | quickly | buy | run |
| **3** | product | love | waste | money | thing | worthless | impressed | waaay | dont | better |

### 4.2.2. KMeans 클러스터링

```
from sklearn.cluster import KMeans
```

```
km = KMeans(n_clusters=4, random_state=1234)
```

```
labels_km = km.fit_predict(tdm)
```

```
top10(labels_km)
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | good | price | quality | product | really | case | far | works | audio | value |
| **1** | great | phone | works | price | product | deal | item | device | worked | battery |
| **2** | phone | product | headset | battery | use | recommend | excellent | quality | work | sound |
| **3** | don | waste | money | buy | time | product | dont | make | mistake | usually |

#### 정규화 후 KMeans

사이킷런의 KMeans는 거리 측정 방식을 바꿀 수 없다. 대신 정규화(normalize) 후 KMeans를 수행하면 코사인 유사도를 사용하는 것과 비슷하게 된다.

```
from sklearn.preprocessing import Normalizer
```

```
nom = Normalizer(copy=False)
```

```
pos = nom.fit_transform(tdm)
```

```
km = KMeans(n_clusters=4, random_state=1234)
```

```
labels_nom = km.fit_predict(tdm)
```

```
top10(labels_nom)
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | good | price | quality | product | really | case | far | works | audio | value |
| **1** | great | phone | works | price | product | deal | item | device | worked | battery |
| **2** | phone | product | headset | battery | use | recommend | excellent | quality | work | sound |
| **3** | don | waste | money | buy | time | product | dont | make | mistake | usually |

### 4.2.3. 스펙트럴 클러스터링

스펙트럴 클러스터링(spectral clustering)은 그래프 기반 클러스터링의 일종이다. 그 핵심 아이디어는 "이웃이 비슷한 것들끼리 묶는다"라고 요약할 수 있다. 스펙트럴 클러스터링은 k-means 등과 달리 군집이 볼록한 형태가 아니라도 가능하다는 장점이 있다.

스펙트럴 클러스터링 알고리즘은 다음과 같다.

1. 각각의 대상으로부터 유사도 그래프를 만든다
2. 라플라시안 행렬로부터 $k$개의 고유벡터를 계산하여 각 대상의 특성으로 삼는다
3. 위의 특성을 바탕으로 k-means 클러스터링을 한다

유사도 그래프(similarity graph)란 비슷한 대상들끼리 연결한 그래프를 말한다. 유사도 그래프를 만드는 방법에는 유사도가 일정 이상인 대상을 연결하는 방법과 가장 유사한 $k$개의 대상을 연결하는 방법이 있다.

라플라시안 행렬(Laplacian matrix)은 대각 원소가 각 꼭지점의 차수(연결된 변의 개수)이고 이외의 원소는 두 꼭지점이 연결되어 있으면 -1이고 그렇지 않으면 0인 행렬이다.

$$L_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

꼭지점 1과 2, 1과 3이 연결되어 있고 2와 3은 연결되어 있지 않은 그래프를 생각해보자. 이 그래프의 라플라시안 행렬은 다음과 같은 형태가 된다.

$$L = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

위의 행렬을 고유값 분해(또는 스펙트럴 분해)를 하면 다음과 같은 고유벡터를 얻는다.

$$V = \begin{bmatrix} .82 & .00 & -.58 \\ -.41 & -.71 & -.58 \\ -.41 & -.71 & -.58 \end{bmatrix}$$

여기서 첫 2개의 고유벡터만 사용한다면 꼭지점 1의 특성은 $(.82, .00)$이고 꼭지점 2와 꼭지점 3은 $(-.41, -.71)$이 된다. 이것은 각 점의 연결 패턴을 바탕으로 차원 축소를 한 것이다.

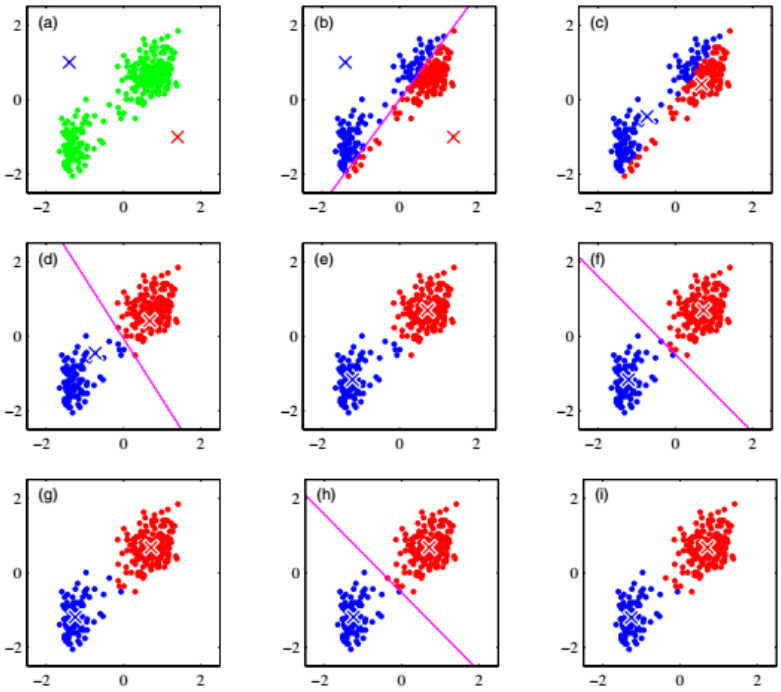
마지막으로 이 특성이 비슷한 점들끼리 k-means로 묶으면 끝난다.

### 4.2.4. K-Means

K-Means는" k개의 평균들"이라는 뜻이다. K-Means는 데이터와 군집의 중심점의 거리를 재서, 그 거리가 가장 가까운 군집에 속한다고 가정한다.

K-Means는 다음과 같은 방식으로 군집을 만든다. 1) k개의 중심점을 무작위로 정한다. 2) 중심점과 거리를 재서 가장 가까운 군집에 모든 데이터를 할당한다. 3) 군집에 속한 멤버들의 평균을 내서 중심점을 정한다. 4) 더이상 중심점에 변화가 없을 때까지 2~3을 반복한다.

아래 그림은 2개의 군집을 가정하고 K-Means를 실시했을 때 군집을 찾아가는 과정이다.




K-Means와 같이 비지도학습에서는 일단 답을 먼저 가정한 다음에 그 가정으로부터 답을 점점 개선시켜 나가는 방법을 많이 쓴다. 다음에 알아볼 가우시안 혼합 모형은 이를 더 일반환시킨 것이다.

## 4.3. 잠재 의미 분석(Latent Semantic Analysis)

단어 문서 행렬은 단어의 의미를 고려하지 않는다. 다음 세 문장을 보자.

1. "고양이 좋다"
2. "고양이 귀엽다"
3. "강아지 좋다"

1번, 2번 문장은 의미가 비슷하고 1번과 3번은 다르지만 단어 문서 행렬에서 거리는 같다. 잠재 의미 분석(Latent Semantic Analsysis)은 이러한 문제를 극복하기 위한 방법이다. LSA는 TDM에 특이값 분해를 적용하여 차원을 축소하는 것이다.

### 4.3.1. 예제 데이터 준비 및 전처리

```
import pandas as pd
from sklearn.externals import joblib
```

```
with open('amazon.pkl', 'rb') as f:
    data = joblib.lboad(f)
locals().update(data)
```

```
tdm.shape
```

```
(1000, 1000)
```

### 4.3.2. LSA 적용

특이값 분해로 차원을 축소한다.

```
from sklearn.decomposition import TruncatedSVD
```

```
svd = TruncatedSVD(n_components=30)
```

`svd.fit_transform()` 에 `tdm` 을 넘겨줘서 차원을 축소한다.

```
pos = svd.fit_transform(tdm)
```

```
pos.shape
```

```
(1000, 30)
```

`pos` 를 확인하면 1000개의 문서가 더 작은 차원으로 축소되었다.

축소된 차원 중에 두 개를 골라 2차원 상에 시각화해보자. 긍정적인 코멘트는 빨간 점으로, 부정적인 코멘트는 파란점으로 나타낸다.

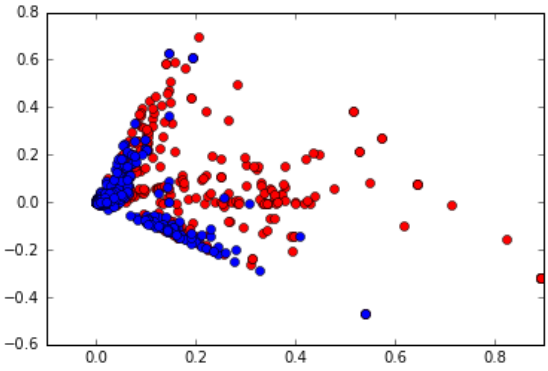
```
import matplotlib.pyplot as plt
```

```
% matplotlib inline
```

```
dim1 = 0
dim2 = 1
plt.plot(pos[sentiment == 1, dim1], pos[sentiment == 1, dim2], 'ro')  # 긍정적인 코멘트 (빨간색)
plt.plot(pos[sentiment == 0, dim1], pos[sentiment == 0, dim2], 'bo')  # 부정적인 코멘트 (파란색)
```

```
[<matplotlib.lines.Line2D at 0x1a23471c908>]
```

```
<matplotlib.figure.Figure at 0x1a23471c940>
```




### 4.3.3. 노멀라이징

LSA로 만들어진 좌표는 문서의 길이에 영향을 받는다. 이 영향을 제거하기 위해 문서의 원점에서 거리를 1로 변환을 해줄 수 있다.

```
from sklearn.preprocessing import Normalizer
```

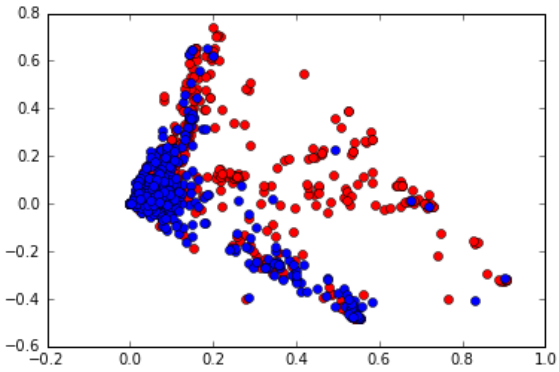
```
norm = Normalizer(copy=False)
```

```
pos2 = norm.fit_transform(pos)
```

```
dim1 = 0
dim2 = 1
plt.plot(pos2[sentiment == 1, dim1], pos2[sentiment == 1, dim2], 'ro')  # 긍정적인 코멘트 (빨간색)
plt.plot(pos2[sentiment == 0, dim1], pos2[sentiment == 0, dim2], 'bo')  # 부정적인 코멘트 (파란색)
```

```
[<matplotlib.lines.Line2D at 0x1a2347ed6d8>]
```

```
<matplotlib.figure.Figure at 0x1a2347ed710>
```




### 4.3.4. 파이프라인 만들기

LSA와 노멀라이징처럼 2단계의 변환이 필요한 경우 파이프라인으로 만들어두면 한 번에 처리를 할 수 있다.

```
from sklearn.pipeline import make_pipeline
```

```
lsa = make_pipeline(svd, norm)
```

```
lsa.transform(tdm)
```

```
array([[ 0.10111665,  0.03605029,  0.0292058 , ...,  0.03278989,
         0.28594578, -0.18861725],
       [ 0.14578546,  0.45944532,  0.27725548, ...,  0.08617946,
        -0.02764001,  0.01591488],
       [ 0.71804303, -0.01378973, -0.52340193, ...,  0.04965575,
        -0.04547243, -0.00981006],
       ...,
       [ 0.06665917,  0.03849417,  0.0271798 , ..., -0.14675044,
         0.07464726,  0.22835009],
       [ 0.05290475,  0.01157312,  0.07038897, ..., -0.10086949,
        -0.58824499,  0.27025315],
       [ 0.07168429,  0.00862645,  0.0413957 , ..., -0.02486053,
         0.10473254,  0.04305728]])
```

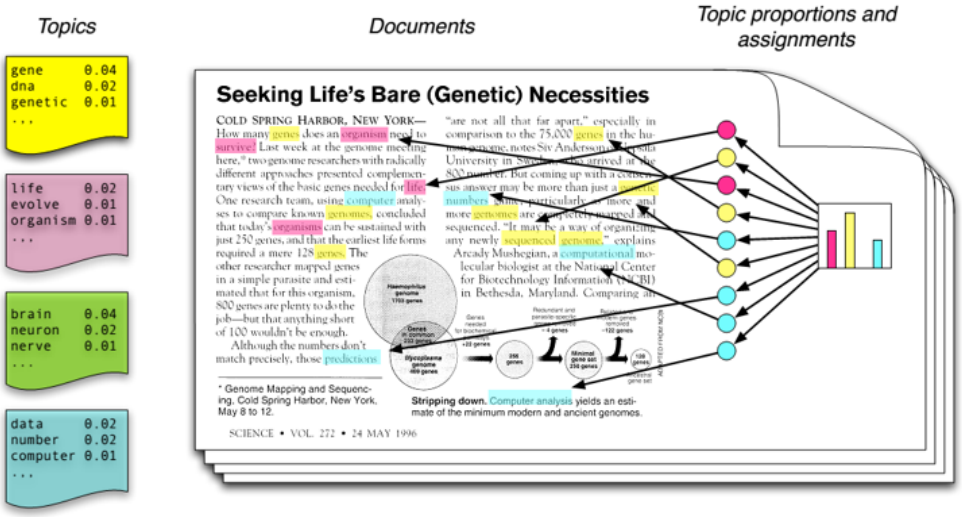
## 4.4. LDA

잠재 디리클레 할당(Latent Dirichlet Allocation, 이하 LDA)은 베이지언 통계학에 바탕을 둔 비지도 텍스트 분석 방법이다. LDA는 2003년에 발표되어 2018년까지 1만번 넘게 인용되었다. LDA의 장점은 분석에 이론적 가정을 추가하기 쉽기 때문에 시간에 따른 변화라든지, 저자의 독특한 개성 등을 포함시켜 분석할 수 있다. 그러나 이러한 분석을 위해서는 더 많은 텍스트들이 필요하고 학습시키기가 까다롭기 때문에 실제로 많이 사용되지는 않는다.

LDA는 다음과 같은 가정에 바탕을 둔다.

- '주제'(topic)는 단어의 분포다
- 각각의 문서는 여러 주제들이 섞여 있다
- 한 문서의 단어들은 문서를 이루는 주제에서 나온 것이다

예를 들면 정치적 주제는 '선거'나 '국회'와 같은 단어들이 나올 확률이 높고, 경제적 주제는 '수요'나 '공급' 같은 단어들이 나올 확률이 높을 것이다. 어떤 글 하나는 정치와 경제가 80:20으로 섞여 있을 수도 50:50으로 섞여 있을 수도 있다. 어떤 주제의 비율이 높냐에 따라 글을 이루는 단어의 비율도 달라지게 된다.




- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of those topics

아래 그림은 뉴욕타임즈 180만개 기사 모음에 LDA를 적용한 결과이다. 각각의 네모칸은 LDA를 통해 발견된 주제를 나타낸다. 네모칸 안의 단어는 해당 주제에서 자주 나오는 단어들을 보여준다. 예를 들면 왼쪽 위의 주제는 'muisc', 'band', 'songs' 등의 단어가 많이 나온다. 즉 음악과 관련된 주제라는 것을 알 수 있다. 사람의 개입 없이 이러한 단어들이 하나의 주제를 이룬다는 것을 컴퓨터가 스스로 발견하는 것이 LDA의 핵심이다.


LDA와 다른 비교사 분석 방법인 클러스터링의 차이를 알아보자. k-means 등 흔히 사용되는 클러스터링에서 하나의 문서는 오직 하나의 클러스터에만 속한다. 반면 LDA는 하나의 문서가 여러 가지 주제를 가질 수 있다.

## 4.4.1. LDA 실습

### gensim 설치

LDA를 위해 `gensim` 을 설치해야 한다.

```
!conda install -y gensim
```

### 데이터 준비

예제 데이터는 인공지능 관련 뉴스메타데이터를 사용한다

```
import pandas as pd
```

```
from sklearn.externals import joblib
```

```
def get_nouns():
    pass
```

```
with open('ai_news.pkl', 'rb') as f:
    data = joblib.load(f)
locals().update(data)
```

`.get_feature_names()` 을 통해 각 차원에 해당하는 단어 목록을 가져오자.

```
words = vectorizer.get_feature_names()
```

단어 목록을 `{단어 번호: 단어}` 형태의 딕셔너리로 변환한다.

```
word_dict = dict(enumerate(words))
```

```
word_dict[0]
```

```
'4차 산업'
```

```
word_dict[100]
```

```
'광주'
```

### gensim 형식으로 변환

`tdm` 을 `gensim` 형식으로 변환한다.

```
from gensim.matutils import Sparse2Corpus
```

`Sparse2Corpus()` 에 `tdm` 의 전치행렬을 넘겨준다.

```
corpus = Sparse2Corpus(tdm.T)
```

```
corpus
```

```
<gensim.matutils.Sparse2Corpus at 0x24fc1537ac8>
```

### 분석

이제 LDA 분석을 위해 `gensim.models` 에서 `LdaModel` 을 불러오자.

```
from gensim.models.ldamodel import LdaModel
```

`LdaModel()` 함수에 몇 가지 옵션을 넘겨준다. `corpus` 옵션에는 생성해놓은 코퍼스를 넘겨준다. `num_topics` 옵션에는 찾는 주제의 수를 넘겨주면 된다. 예제에서는 100가지 주제를 찾도록 지정하자. 기존에 생성한 인덱스와 단어로 이루어진 단어 사전인 `word_dict` 를 `id2word` 옵션에 넘겨준다.

`passes` 와 `iterations` 는 계산 횟수를 정해준다. 기본값은 각각 `1` 과 `50` 이나 증가시켜주면 결과가 더 안정적으로 나온다.

`LdaModel()` 은 계산에 랜덤한 부분이 있기 때문에 `random_state` 를 고정시켜 계산 결과를 일정하게 만들어준다.

```
lda = LdaModel(corpus=corpus,
               num_topics=100,
               passes=3,
               iterations=100,
               id2word=word_dict,
               random_state=123)
```

### 토픽별 단어 확인

`.show_topic()` 을 사용하면 각 토픽의 단어 분포를 확인할 수 있다.

```
lda.show_topic(0)
```

```
[('뉴스', 0.21745048),
 ('보유', 0.17862064),
 ('가격', 0.113770254),
 ('언어', 0.09600149),
 ('응용', 0.068983026),
 ('해결', 0.055789713),
 ('매체', 0.05529951),
 ('추세', 0.044599134),
 ('인공지능', 0.034755576),
 ('전문', 0.032504205)]
```

```
lda.show_topic(2)
```

```
[('개최', 0.24177672),
 ('대회', 0.16730691),
 ('일보', 0.10940601),
 ('디자인', 0.09314068),
 ('대전', 0.08759076),
 ('중도', 0.08741614),
 ('구체', 0.03957544),
 ('충북', 0.037207205),
 ('지역', 0.032445055),
 ('설명회', 0.029183898)]
```

처음 `LdaModel()` 을 생성할 때, `random_state` 옵션을 지정하지 않고, LDA 를 다시 실행해보면, 다른 결과가 도출될 수 있다.

### 문서의 주제 확인

#### 문서 변환

이번에는 문서에 어떤 비율로 주제가 섞여있는지 확인해보자.

첫 번째 문서의 단어 빈도수를 확인해보자. 단어 번호와 첫 번째 문서에서의 빈도수로 이루어진 리스트를 생성하자.

```
row = tdm[0]
doc = list(zip(row.indices, row.data))
doc
```

```
[(897, 1),
 (995, 1),
 (637, 1),
 (564, 1),
 (990, 1),
 (522, 1),
 (24, 1),
 (673, 1),
 (562, 1),
 (45, 2),
 (466, 1),
 (759, 2),
 (541, 1),
 (776, 1)]
```

단어 목록에서 단어 번호를 단어로 바꿔보자

```
doc_words = [(words[i], n) for i, n in doc]
```

```
doc_words
```

```
[('특징', 1),
 ('효과', 1),
 ('인공지능', 1),
 ('완성', 1),
 ('활용', 1),
 ('엔진', 1),
 ('개발', 1),
 ('자체', 1),
 ('올해', 1),
 ('게임', 2),
 ('시작', 1),
 ('준비', 2),
 ('열풍', 1),
 ('지난해', 1)]
```

**문서 토픽 확인하기**

이번에는 첫 번째 문서의 **토픽 비율**을 확인하자. `lda.get_document_topics()` 에 `doc` 를 넘겨준다.

```
lda.get_document_topics(doc)
```

```
[(12, 0.063232705),
 (22, 0.15562035),
 (28, 0.17648429),
 (39, 0.115683466),
 (44, 0.1592615),
 (52, 0.087478064),
 (87, 0.119961806),
 (96, 0.06816017)]
```

`.show_topic()` 을 이용하여 해당 토픽에 어떤 단어들이 분포해 있는지 확인할 수 있다.

```
lda.show_topic(28)
```

```
[('대표', 0.21936047),
 ('플랫폼', 0.17928517),
 ('모바일', 0.15007903),
 ('인공지능', 0.06143891),
 ('게임', 0.05209241),
 ('조사', 0.03942193),
 ('자사', 0.03885676),
 ('전체', 0.034782927),
 ('기반', 0.03356984),
 ('국내', 0.02857214)]
```

### 모형 저장

이제 분석한 LDA 모형을 저장해보자. `lda.save()` 에 파일명을 넘겨주면 해당 이름으로 저장 된다. `'lda_test.lda'` 라는 파일명으로 저장해보자.

```
lda.save('lda_test.lda')
```

### 모형 불러오기

저장한 lda 모형을 불러올 때, `LdaModel.load()` 를 사용하면 된다.

```
lda2 = LdaModel.load('lda_test.lda')  # 기존 모형과 구분짓기 위해 lda2 로 저장한다.
```