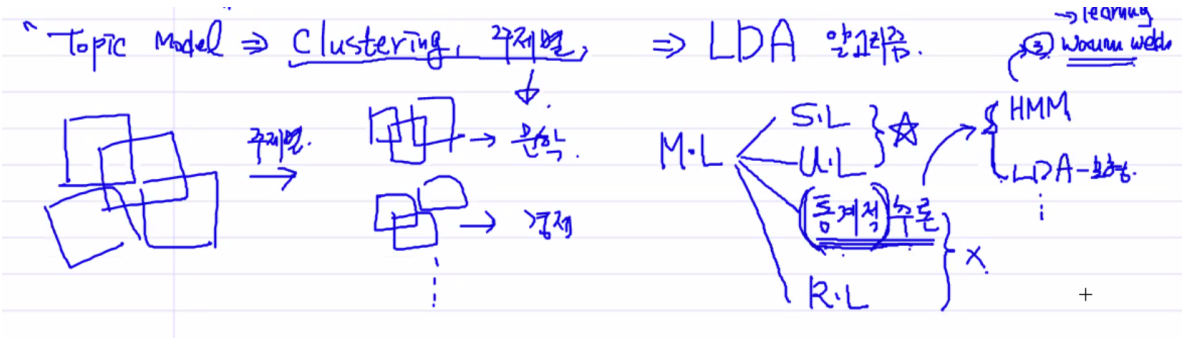
출처가 명시되지 않은 모든 자료(이미지 등)는 조성현 강사님 블로그 및 강의 자료 기반.

토픽 모델

Topic Modeling(주제 식별)이란, 문서 집합에 숨어 있는 주제를 찾아내는 텍스트 마이닝 기법이다. 여러 개의 문서(문장)가 있을 때, 그 문서들을 주제별로 클러스터링하는 방법이다. 클러스터링하기 때문에 당연히 머신러닝의 범주에 속하며, 다음의 기법들을 활용해 진행할 수 있다.

- Supervised Learning
- Unsupervised Learning
- 통계적 추론
  - HMM
  - 바움웰치
- Reinforcement Learning



주로 LSA, LDA 기법을 많이 사용하며, 그 중에서도 특히 LDA 알고리즘이 더 최신 기법이기 때문에 많이 사용한다.

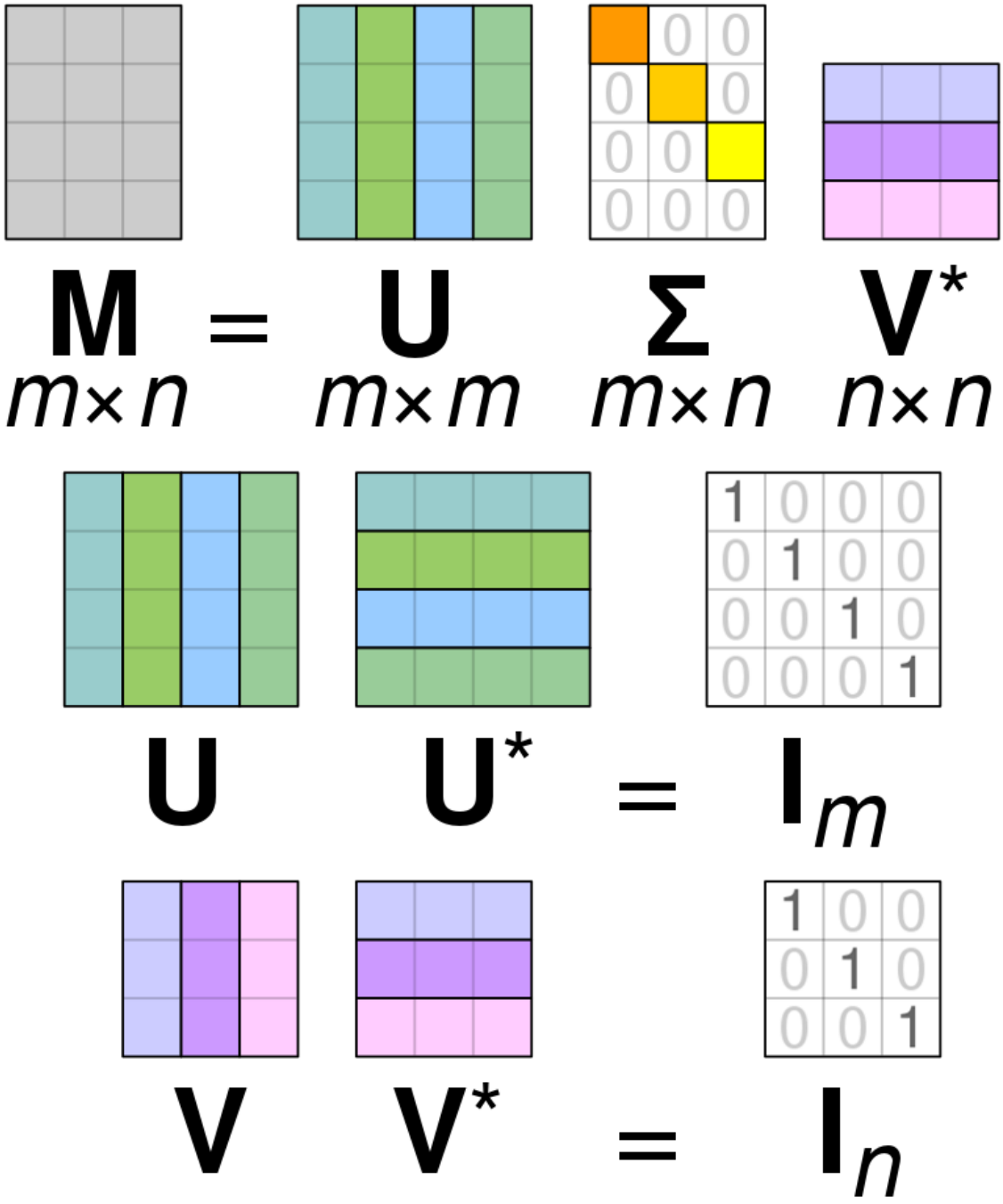
1. LSA(*Latent Semantic Analysis*)

가장 기본적인 토픽 모델이다. 차원축소를 위해 사용하는 선형대수학의 대표적인 기법으로 `주성분분석(PCA)`과 `특이값분해(SVD)`가 있다. 그 중에서 후자의 개념을 바탕으로 문서 데이터의 의미를 축소하고, 잠재된 의미를 분석하는 기법이다.

문서 데이터를 분해하기에 앞서, 기본적으로 문서 데이터는 수치화되었다고 가정한다. 앞으로 언급하는 **문서** 데이터라 함은, 말 그대로의 문장이 아니라, 문장을 앞서 배운 TF-IDF 모델을 바탕으로 수치화시켜 놓은 데이터를 의미한다.

SVD

특이값 분해(Singular Value Decomposition)이란, 선형대수학에서 $m * n$ 크기의 행렬을 특이 벡터로 이루어진 행렬로 분해하는 과정이다.



출처: 위키피디아 singular value decomposition

특이값 분해의 결과인 우변의 첫 번째 행렬 U와 마지막 행렬 V^T (위의 그림과 첨자가 다르지만, 강사님과 같은 표현을 따라 이렇게 표기한다.) 가 **직교행렬**이라는 것이 중요하다. 직교행렬의 경우, 행렬을 이루는 모든 열벡터의 내적이 0이 되는 성질을 갖는다.

SVD의 결과로 나온 대각행렬 Σ 의 대각 원소들이 특이값이다. 이 대각행렬을 제곱하면 그 결과는 행렬 C의 제곱과 같아진~~다(C/고 한)~~다. 원래 행렬의 특징을 잘 나타내는 고윳값들로 이루어진 사상이라고 생각하자. 이 때 Σ 의 대각 원소들은 우하향하는 방향으로 갈수록 작아지게 정렬된다.

LSA와의 연계

$C = U \cdot S \cdot V^T$

U 4 x 4

```
[[-0.65  0.    0.27 -0.71]
 [-0.31 -0.59 -0.74  0.  ]
 [-0.65  0.    0.27  0.71]
 [-0.23  0.8  -0.55 -0.  ]]
```

$\cdot$

S 4 x 8

```
[[1.49 0.   0.   0.   0.   0.   0.   0.  ]
 [0.   1.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.89 0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.  ]]
```

$\cdot$

V^T 8 x 8

```
[[-0.14 -0.07 -0.14 -0.07 -0.07 -0.67 -0.07 -0.71]
 [-0.38  0.38 -0.38  0.38  0.38  0.24  0.38 -0.24]
 [-0.54 -0.29 -0.54 -0.29 -0.29  0.25 -0.29  0.09]
 [-0.74  0.03  0.62  0.03  0.03 -0.19  0.03  0.19]
 [-0.   -0.16 -0.03 -0.49  0.84 -0.04 -0.16  0.04]
 [-0.    0.66  0.2  -0.52 -0.17  0.32 -0.17 -0.32]
 [-0.   -0.16 -0.03 -0.49 -0.16 -0.04  0.84  0.04]
 [-0.    0.53 -0.35 -0.11 -0.04 -0.54 -0.04  0.54]]
```

문서 데이터 C를 우변의 U, S, V^T로 분해했다. 원래 문서 데이터 C는 4개의 문서(Document)로부터 빈도가 높은 8개의 단어만을 사용하여 수치화한 TF-IDF 행렬이다. 따라서 C의 shape은 `(4, 8)` 이 된다.

> *참고* : TF-IDF 행렬의 shape
>
> 앞의 수업에서는 행에 단어를 놓고 열에 문서를 놓았다. 그러나 Scikit-learn에서 `TfidfVectorizer` 함수를 사용하면 행에 문서가 오고 열에 단어가 온다. 따라서 `→`의 방향으로 행렬을 해석해야 하며, 이렇게 해석할 때 각 *행*은 하나의 **문서**가 되고, *열*은 그 문서에 사용된 **단어의 중요도**를 나타낸 것이다.

특이값 분해 후 나온 우변의 결과를 보자. 먼저 U는 `(4, 4)`의 행렬이다. 이 행렬은 직교행렬이므로, 각 문서 4개를 나타내는 벡터끼리의 내적이 0이 된다. 문서를 서로 직교하게 만들어 놓음으로써, 각 문서 간 상관성을 0으로 맞춰준 것이다.

다음으로 S는 `(4, 8)`의 대각행렬이다. *해석*의 문제가 들어 온다. U와 S를 곱한 행렬은 원래 데이터에서 중요한 의미가 무엇인지를 나타내는 데이터가 된다. LSA에 적용하기 위해서는 이것을 **의미**로서 해석한다. 즉, 문서가 어떠한 의미군에 속해 있는지를 나타낸다.

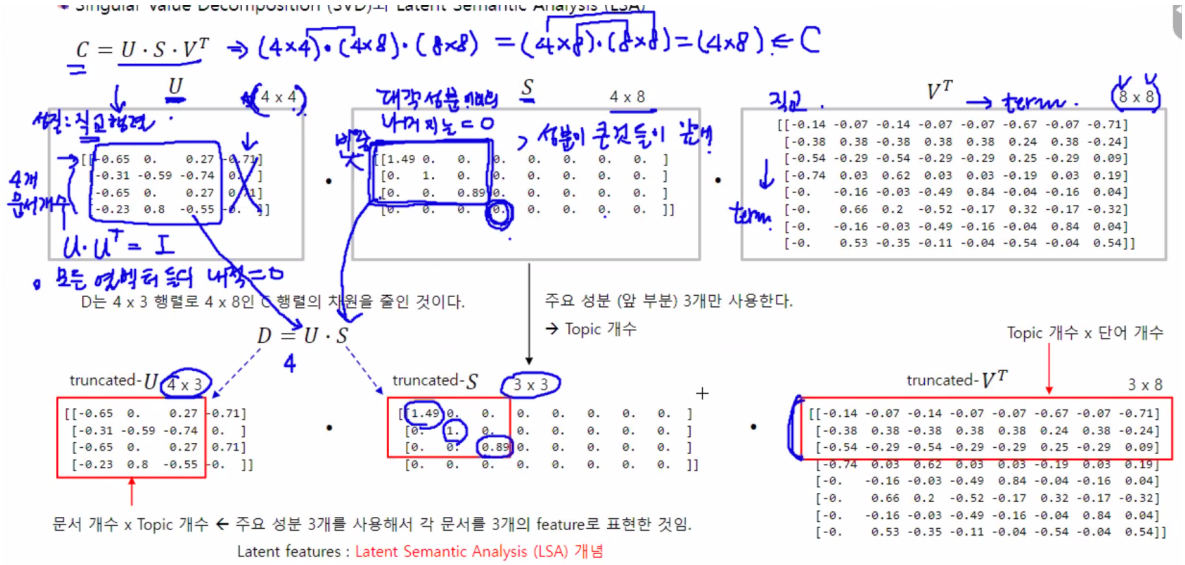
마지막으로 V^T는 전치된 직교 행렬로, 해당하는 의미군에 어떤 단어들이 많이 사용되었는지를 나타낸다.

Truncated SVD

LSA에서는 trauncated SVD만 사용하여 의미 있는 몇 가지의 주제만을 추출하고자 한다. *truncated*의 의미란, 대각행렬 S의 크기를 제한하고, 양 옆 직교 행렬들의 크기를 제한한 대각행렬의 크기에 맞추는 방법이다.

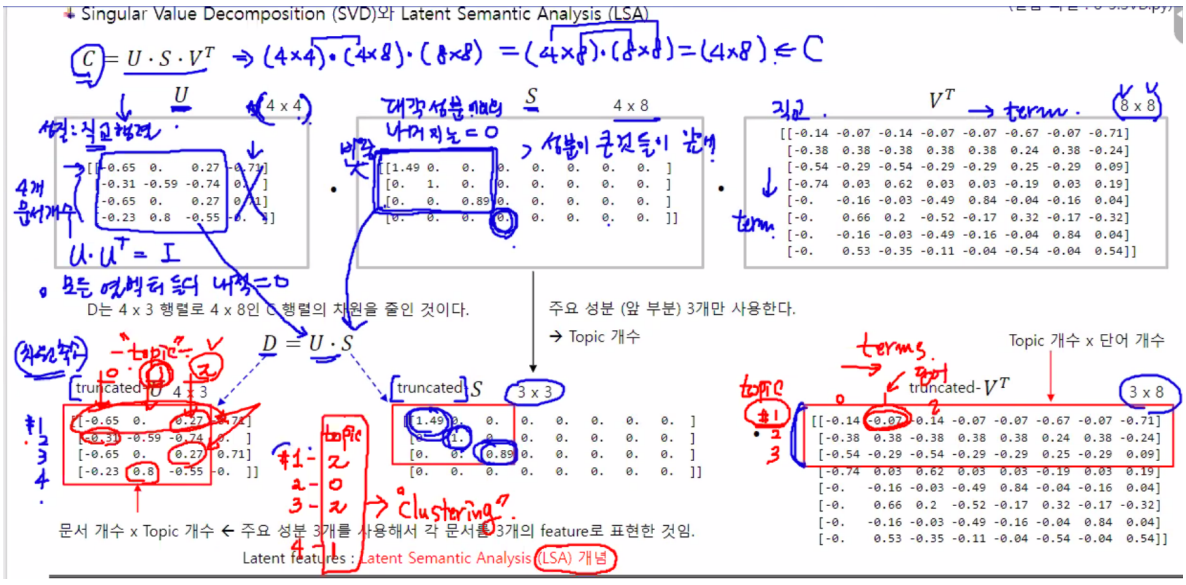
S가 큰 수치일수록 앞 쪽에 배치되고, $U \cdot S$가 주제를 나타낸다고 해석하기 때문에, 앞 쪽에서부터 큰 몇 개의 수치를 선택하면 중요한 몇 개의 주제만을 보기 위함이라 해석할 수 있다. 즉, 문서의 의미를 담고 있는 *latent feature*이며, 그 중에서도 상위의 몇 가지를 선택하는 것이다.

다음의 예를 확인하자.



S에서 대각성분의 값이 0이 아닌 것만 선택하여 중요도가 높은 3개의 feature를 선택한다. 행렬곱의 원리에 맞게 U, V^T의 shape도 맞추어 선택해 준다. *truncated* U는 `(4, 3)`의 행렬, *truncated* V^T는 `(3, 8)`의 행렬이 된다.

> 질문? US 곱한거에서 큰 거 뽑아와야 하는거 아닌가?
>
> ㅇㅇㅇㅇ 그러하답니다!~



4개의 문서들 각각의 feature가 줄어 든다. 원래 문서 데이터를 나타내는 행렬 C가 $D = U \cdot S$가 되면서 `(4, 3)`의 행렬로 크기가 줄어든다. 문서의 의미 정보 중 중요한 것만 반영되도록 차원을 선택하여 축소한 것이다. D에서 숫자가 클수록 해당 topic일 확률이 높다고 본다. 이렇게 topic별로 문서를 클러스터링한 것이다.

다시 말해, D에서 첫 번째 feautre가 큰 묶음이 하나의 집합으로 클러스터링되고, 그러한 의미에서 이것을 topic이라고 해석하는 것이다. 여기서는 3개의 topic을 선택했기 때문에, 각 문서가 3개의 주제 집합으로 클러스링된다.

맨 오른쪽의 행렬 V^T에서는 행이 곧 topic의 개수를, 열이 문서 데이터에 사용된 단어(term)를 나타낸다. topic1에서 가장 큰 숫자를 뽑았을 때 그 열에 해당하는 단어가 topic1에서의 빈출(?) 단어라는 의미이다.

실습

카테고리별로 분류된 뉴스 데이터를 LSA에 의해 주제별로 크러스터링 해보자.

필요한 모듈은 아래와 같다. 뉴스 데이터는 강사님이 저장해 두신 `pickle` 형태의 데이터를 사용한다.

```
# 모듈 불러 오기
import numpy as np
import re
import pickle
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

# 경로 설정
root_path = "/content/drive/My Drive/멀티캠퍼스/[혁신성장] 인공지능 자연어처리 기반/[강의]/조성현 강사님"
data_path = f"{root_path}/dataset"
```

뉴스 데이터를 불러 오고, 20개의 카테고리를 확인해 보자.

```
# 데이터 로드
with open(f"{data_path}/news.data", 'rb') as f:
    news_data = pickle.load(f)

# 데이터 조회 및 target 확인
news = news_data.data
print(len(news)) # 총 11314개의 뉴스 데이터
print(news[0])
print(news_data.target_names)
```

```
['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']
```

데이터를 전처리한다.

- 영문자 제외 모두 제거.
- 영어 소문자 변환.
- 영어 불용어 제거.
- 단어 길이 3 이상만 사용.

```
# 전처리
news_1 = []
for doc in news:
    news_1.append(re.sub("[^a-zA-Z]", " ", doc)) # 영문자

news_2 = []
stopwords_list = stopwords.words('english')
for doc in news_1:
    temp = []
    for w in doc.split():
        w = w.lower() # 소문자
        if len(w) > 3 and w not in stopwords_list: # 단어 길이, 불용어
            temp.append(w)
    news_2.append(" ".join(temp))
```

전처리된 데이터를 가지고 Tf-Idf 행렬을 만든다. 단어 출현 빈도를 최소 500으로 제한한다.

- `.vocabulary_` : 어휘집 확인

```
# Tf-Idf 행렬
tf_vector = TfidfVectorizer(max_features=500) # 빈도 제한
tf_vector.fit(news_2)
print(tf_vector.vocabulary_) # 어휘 목록 확인
tfidf_matrix = tf_vector.transform(news_2)
```

Truncated SVD를 진행한다. Tf-Idf 행렬에 대해 진행하며, 20개 feature만 선택한다. `n_iter` 값을 충분히 주지 않으면 계산이 제대로 되지 않을 수 있다.

- `.singular_values` : 특잇값 확인. 대각행렬로 만들어 주어야 함.
- `.components` : V^T에 해당.

```
svd = TruncatedSVD(n_components=len(news_data.target_names), n_iter=1000)
svd.fit(tfidf_matrix)

# 특잇값분해 결과
U = svd.fit_transform(tfidf_matrix) / svd.singular_values_
S = np.diag(svd.singular_values_)
VT = svd.components_
```

U의 shape을 확인하면, `문서 개수 x topic 개수` 만큼의 원소가 있음을 확인할 수 있다. 그리고 U에서 가장 큰 인덱스를 찾으면, 그것이 바로 각 문서별로 가장 연관성이 높은 주제가 된다.

> *질문*
>
> $U \cdot S$에서 찾아와야 하는 것 아닌가? 그런 듯! 코드 수정 필요합니다. `np.dot(U, S)`

```
# 문서별 Topic 번호 확인: U에서 가장 큰 feature 인덱스 찾기
for i in range(15): # 문서 15개만 확인
    print('문서 = {:d} : Topic = {:d}'.format(i, np.argmax(U[i:(i+1), :][0])))
```

```
문서 = 0 : Topic = 17
문서 = 1 : Topic = 0
문서 = 2 : Topic = 17
문서 = 3 : Topic = 5
문서 = 4 : Topic = 3
문서 = 5 : Topic = 8
문서 = 6 : Topic = 0
문서 = 7 : Topic = 6
문서 = 8 : Topic = 7
문서 = 9 : Topic = 12
문서 = 10 : Topic = 1
문서 = 11 : Topic = 10
문서 = 12 : Topic = 9
문서 = 13 : Topic = 0
문서 = 14 : Topic = 6
```

V^T를 확인하면 토픽별 주요 단어를 확인할 수 있다. 우선 10개씩만 확인해 보자.

- `np.flipud` : 위 아래로 뒤집는다.
- `.argsort()` : 작은 값부터 인덱스의 위치를 반환한다.

```
for i in range(len(VT)):
    idx = np.flipud(VT[i].argsort())[:10]
    print('토픽-{:2d} : '.format(i+1), end='')
    for i in idx:
        print('{:s} '.format(vocab[n]), end='')
    print()
```

```
토픽- 1 : would like know people think good also could time well
토픽- 2 : thanks windows please anyone mail card know advance drive file
토픽- 3 : would thanks anyone know like please could mail someone advance
토픽- 4 : game team year games good last season players play hockey
토픽- 5 : would like drive system windows card scsi disk team problem
토픽- 6 : drive please scsi hard mail sale would email drives people
토픽- 7 : drive know like anyone scsi drives hard something card think
토픽- 8 : like please sale mail email offer something send list interested
토픽- 9 : think windows people please card jesus thanks believe bible mail
토픽-10 : good card think sale price bike also much looking offer
토픽-11 : card people video know sale monitor government drivers price offer
토픽-12 : think chip system could encryption clipper need government space much
토픽-13 : could thanks right card problem much bike well someone advance
토픽-14 : good people windows file government files thanks drive would year
```

```
토픽-15 : anyone thanks like good also people space card could system
토픽-16 : space year thanks problem nasa much bike also time think
토픽-17 : problem thanks system game need window jesus time first using
토픽-18 : anyone right israel think sale window problem israeli government back
토픽-19 : problem people please anyone could system email problems good time
토픽-20 : also year good know window problem please israel last using
```

최종적으로 문서별로 분류된 topic의 코드를 확인하면 다음과 같다.

```
def check_topic(x, y):
    print("문서 %d의 topic = %s" %(x,
news_data.target_names[news_data.target[x]]))
    print("문서 %d의 topic = %s" %(y,
news_data.target_names[news_data.target[y]]))

check_topic(1, 6)
check_topic(0, 2)
```

```
문서 1의 topic = alt.atheism
문서 6의 topic = comp.sys.mac.hardware
문서 0의 topic = talk.politics.mideast
문서 2의 topic = talk.politics.mideast
```