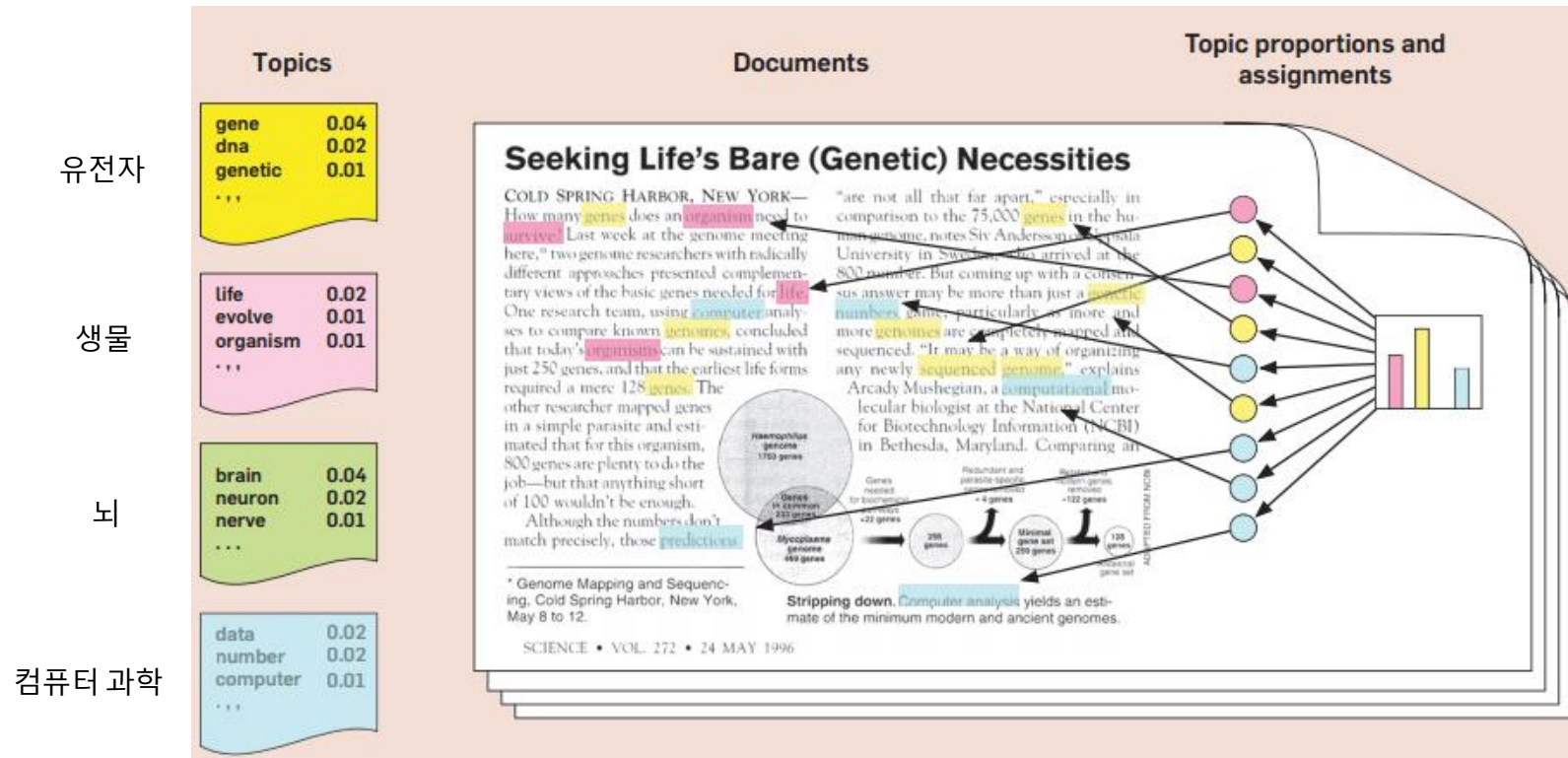


TOPIC MODELING

토픽 모델링(Topic Modeling)

- 구조화되지 않은 방대한 문헌집단에서 주제를 찾아내기 위한 알고리즘
 - 의미가 유사한 단어들을 클러스터링(그룹화)하여 주제를 추론(생성)하는 모델



proportion: 비율
assignment: 배치, 배정

토픽 모델링 종류

- **LDA(Latent Dirichlet Allocation)**

- 문서들은 토픽들의 혼합으로 구성되어 있고, 토픽들은 확률 분포에 기반하여 단어들을 생성한다고 가정함. 데이터가 주어지면, 문서가 생성되는 과정을 역추적하는 방식임

- **ATM(Author Topic Model)**

- LDA는 문서별 주제분포를 계산하는 반면, ATM은 **저자별 주제분포**를 계산함

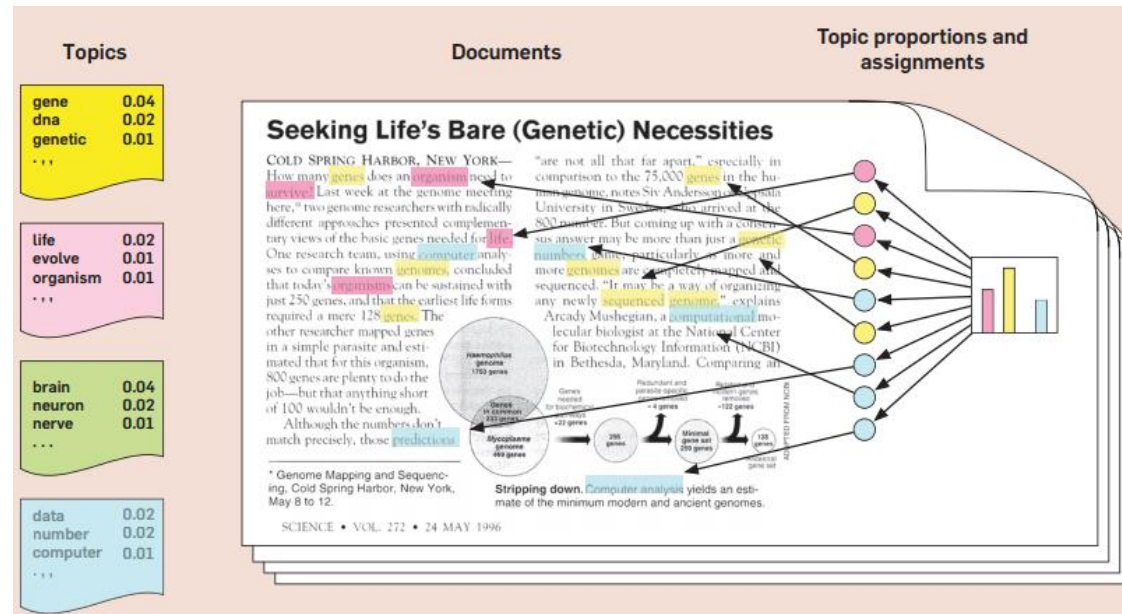
- **DMR(Dirichlet Multinomial Regression)**

- LDA를 확장한 기법으로 **문서의 메타데이터(저자, 연도, 발행처)에 따른 주제 분포를 계산할 수 있음**

토픽 모델링 종류

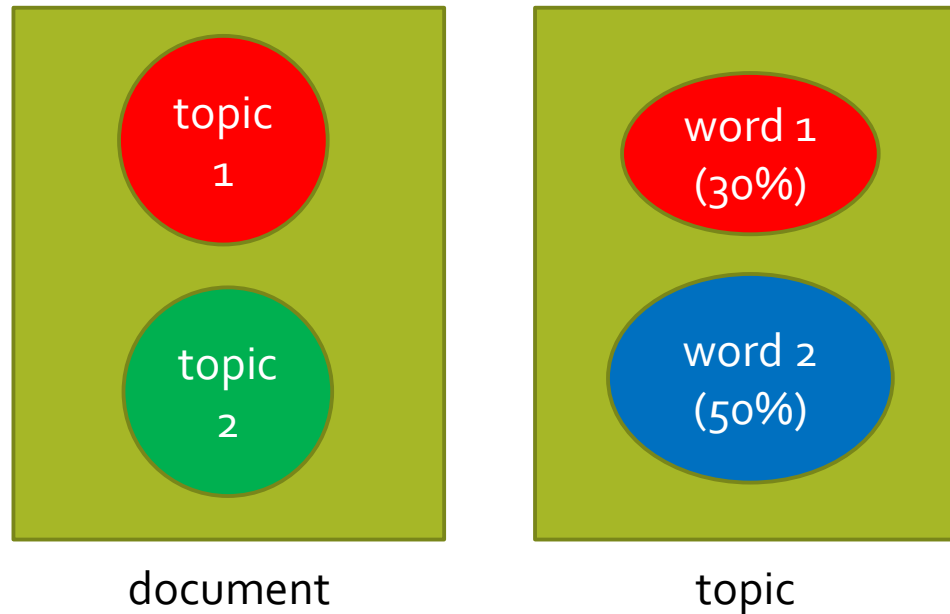
- **LDA(Latent Dirichlet Allocation)**

- **Latent:** 잠재적인, **Dirichlet:** 수학적 이름, **Allocation:** 할당
- 문서들은 토픽들의 혼합으로 구성되어 있고, 토픽들은 확률 분포에 기반하여 단어들을 생성한다고 가정함
- 데이터가 주어지면, 문서가 생성되는 과정을 **역추적**하여 토픽을 찾아나가는 방법



LDA(Latent Dirichlet Allocation)

- 잠재 디리클레 할당(LDA)은 토픽 모델링의 대표적인 알고리즘
- 코드 없이 실습: <https://lettier.com/projects/lda-topic-modeling>
- 문서들은 토픽들의 혼합으로 구성되어 있고, 토픽들은 확률 분포에 기반하여 단어들을 생성한다고 가정함



LDA(Latent Dirichlet Allocation)

- 문서가 작성되는 (잠재적인) 과정
 - 1) 문서에 사용할 단어의 개수 **N**을 정함
 - 5개의 단어 선택: 뼈다귀, 말티즈, 생선, 야옹이, 집사
 - 2) 문서에 사용할 토픽의 혼합을 결정
 - 토픽을 2개라고 정했으면 강아지 관련된 토픽 **60%**, 고양이 관련 토픽 **40%**을 정함
 - 3) 문서에 사용할 각 단어를 아래와 같이 정함
 - 3-1) 토픽 분포에서 토픽 **T**를 확률적으로 선택
 - 2)에서 정한 토픽 확률을 기반하여 **60%** 강아지, **40%** 고양이 토픽 선택
 - 3-2) 선택한 토픽 **T**에서 단어의 출현 확률 분포에 기반해 문서에 사용할 단어를 고름
 - 강아지 토픽이 선택되었다면, 강아지 토픽내에서 단어별로 확률을 기반하여 선택
 - Ex) 강아지 토픽인 경우, 강아지: **30%**, 뼈다귀: **20%**, 말티즈: **30%**, ...

LDA(Latent Dirichlet Allocation)

- LDA 알고리즘

- 1) 사용자는 **토픽의 개수 k** 를 제공
 - 토픽 k 가 M 개의 문서에 걸쳐 분포되어 있다고 가정
- 2) 모든 단어를 k 개 중 하나의 토픽에 각각 무작위로 할당
- 3) 모든 문서내의 모든 단어에 대해 아래의 알고리즘을 반복 진행
 - 3-1) 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 따라 토픽이 재할당
 - $p(\text{topic } t \mid \text{document } d)$: 문서 d 의 단어들 중 토픽 t 에 해당하는 단어의 비율
 - $p(\text{word } w \mid \text{topic } t)$: 단어 w 를 갖고 있는 모든 문서들 중 토픽 t 가 할당된 비율

LDA(Latent Dirichlet Allocation)

- 단어 w 는 아래의 두 기준에 따라 토픽을 재할당
 - 1. $p(\text{topic } t \mid \text{document } d)$: 문서 d 의 단어들 중 토픽 t 에 해당하는 단어들의 비율
 - 2. $p(\text{word } w \mid \text{topic } t)$: 단어 w 를 갖고 있는 모든 문서들 중 토픽 t 가 할당된 비율
- Ex) doc1의 apple의 토픽을 할당한다면?

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

LDA(Latent Dirichlet Allocation)

- Ex) doc1의 apple의 토픽을 할당한다면?
 - 1. $p(\text{topic } t \mid \text{document } d)$: 문서 d 의 단어들 중 토픽 t 에 해당하는 단어들의 비율
 - doc1의 모든 단어들은 토픽 A와 토픽 B 50대 50의 비율로 할당됨
 - => 기준 1을 가지고 doc1 apple의 토픽을 할당할 수 없음

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

LDA(Latent Dirichlet Allocation)

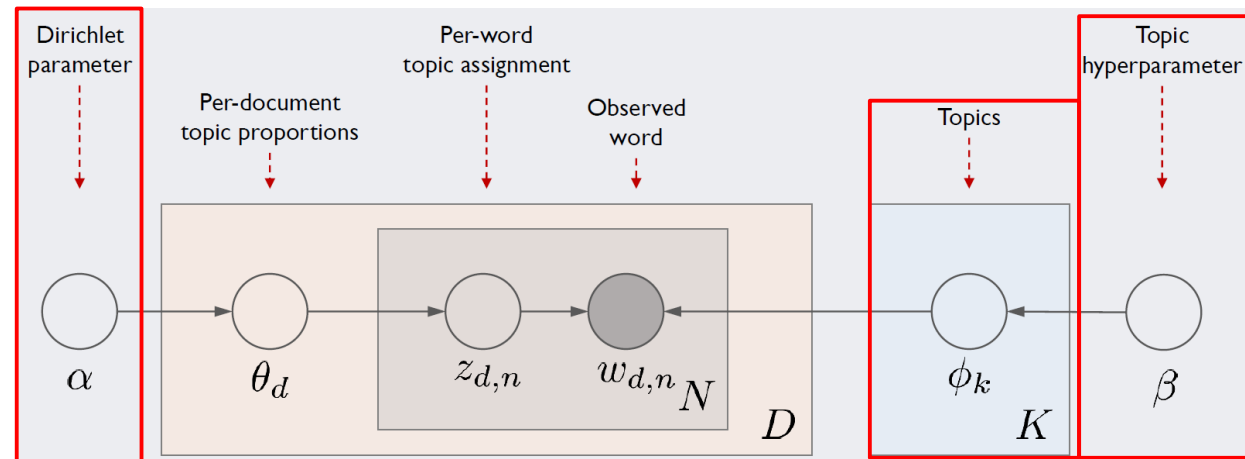
- Ex) doc1의 apple의 토픽을 할당한다면?
 - 2. $p(\text{word } w \mid \text{topic } t)$: 단어 w 를 갖고 있는 모든 문서들 중 토픽 t 가 할당된 비율
 - apple이 전체 문서에서 어떤 토픽에 할당되어 있는지를 관찰
 - => 기준 2를 가지고 LDA는 doc1의 apple을 B 토픽에 할당할 수 있음

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

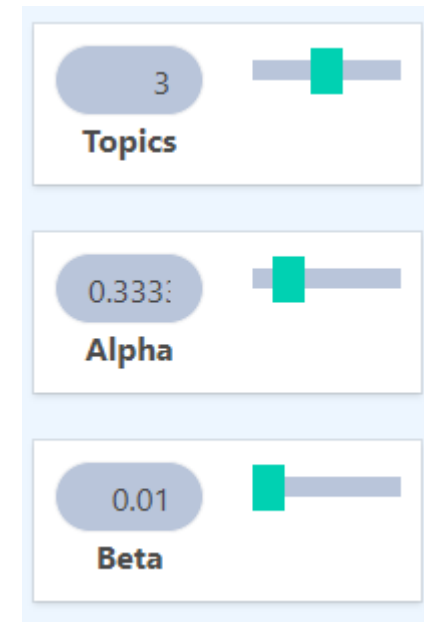
LDA(Latent Dirichlet Allocation)

- 모델 아키텍처
 - **D**: 말뭉치 전체 문서 개수
 - **K**: 전체 토픽 수
 - **N**: **d**번째 문서의 단어 수
- 하이퍼 파라미터: α, β, K
 - 하이퍼 파라미터는 모델의 성능에 영향을 줄 수 있는 매개변수로 분석가가 정해야 됨



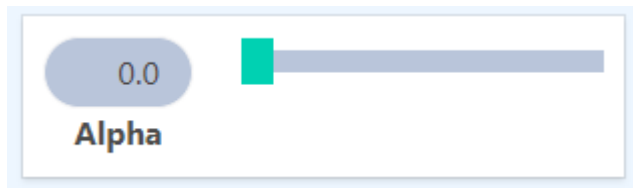
LDA(Latent Dirichlet Allocation)





- 하이퍼 파라미터
 - 모델의 성능에 영향을 줄 수 있는 매개변수로 분석가가 정해야 됨
 - α : 문서-토픽분포를 위한 Hyper-parameter
 - 작을수록 적은 수의 토픽으로 쏠림
 - β : 토픽-단어분포를 위한 Hyper-parameter
 - 작을수록 적은 수의 단어로 쏠림
 - K : 전체 토픽 수
 - <https://lettier.com/projects/lda-topic-modeling>





LDA(Latent Dirichlet Allocation)

- 하이퍼 파라미터
 - α : 문서-토픽분포를 위한 Hyper-parameter
 - 작을수록 **토픽**이 한 쪽으로 쏠림
 - α 가 작은 경우



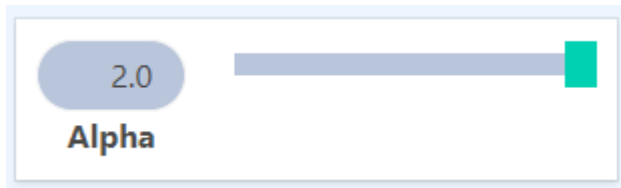
		
Document 0	20	25
Document 1	1	44
Document 2	26	19
Document 3	9	36
		





	Topic 0	Topic 1	Topic 2
	0.511	0.213	0.043
	0.489	0.787	0.957
	Topic 0	Topic 1	Topic 2



	Topic 0	Topic 1	Topic 2
Document 0	1.000	0.000	0.000
Document 1	0.000	0.000	1.000
Document 2	1.000	0.000	0.000
Document 3	0.000	1.000	0.000
	Topic 0	Topic 1	Topic 2

LDA(Latent Dirichlet Allocation)

- 하이퍼 파라미터
 - α : 문서-토픽분포를 위한 Hyper-parameter
 - 클수록 **토픽**이 여러 개로 쉰림
 - α 가 큰 경우



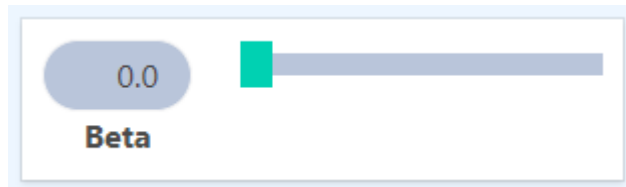
		
Document 0	20	25
Document 1	1	44
Document 2	26	19
Document 3	9	36
		





	Topic 0	Topic 1	Topic 2
	0.057	0.964	0.023
	0.943	0.036	0.977
	Topic 0	Topic 1	Topic 2

	Topic 0	Topic 1	Topic 2
Document 0	0.333	0.392	0.275
Document 1	0.333	0.059	0.608
Document 2	0.412	0.529	0.059
Document 3	0.745	0.216	0.039
	Topic 0	Topic 1	Topic 2

LDA(Latent Dirichlet Allocation)

- 하이퍼 파라미터
 - β : 토픽-단어분포를 위한 Hyper-parameter
 - 작을수록 단어가 **한 쪽 토픽으로** 쏠림
 - β 가 작은 경우









		
Document 0	20	25
Document 1	1	44
Document 2	26	19
Document 3	9	36
		
		</

LDA(Latent Dirichlet Allocation)

- 하이퍼 파라미터
 - β : 토픽-단어분포를 위한 Hyper-parameter
 - 클수록 단어들이 여러 개의 토픽으로 쏠림
 - β 가 큰 경우



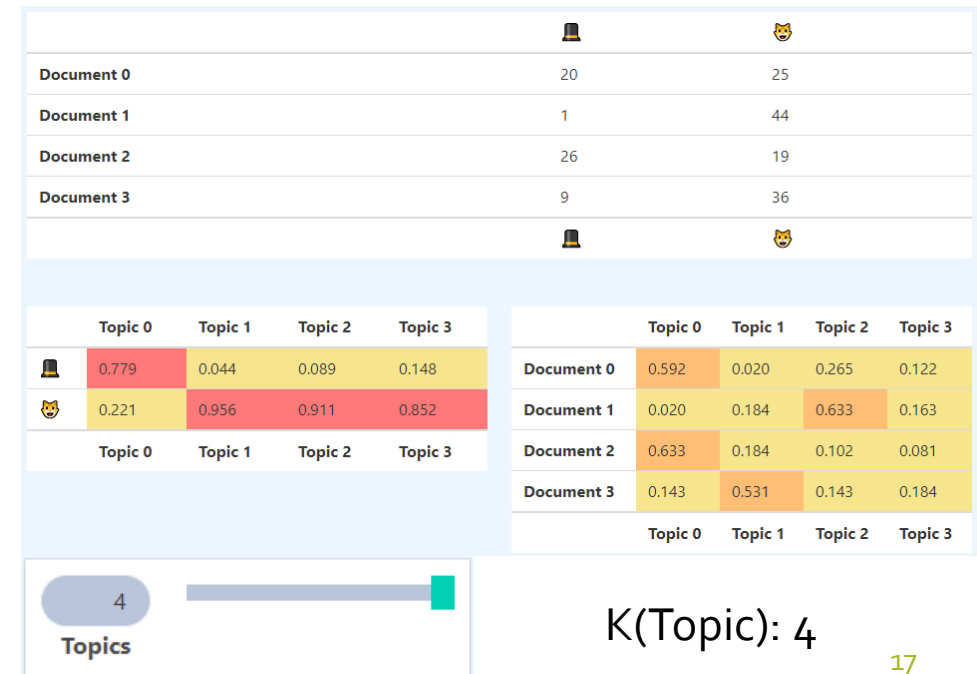
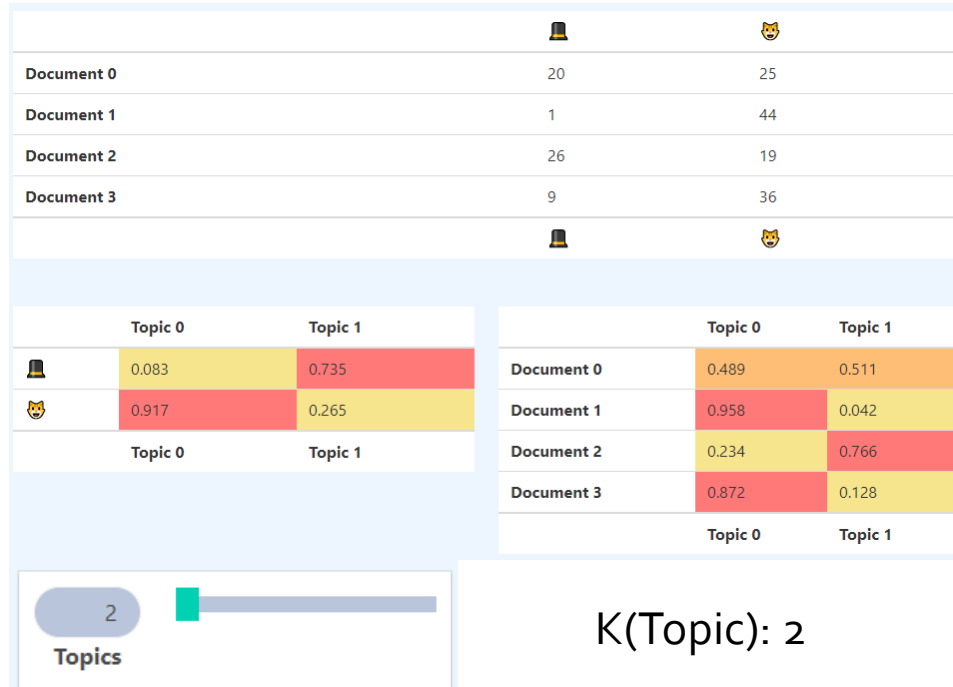
		
Document 0	20	25
Document 1	1	44
Document 2	26	19
Document 3	9	36
		

	Topic 0	Topic 1	Topic 2
	0.068	0.743	0.435
	0.932	0.257	0.565
	Topic 0	Topic 1	Topic 2

	Topic 0	Topic 1	Topic 2
Document 0	0.083	0.083	0.834
Document 1	0.938	0.041	0.021
Document 2	0.083	0.521	0.396
Document 3	0.729	0.083	0.187
	Topic 0	Topic 1	Topic 2

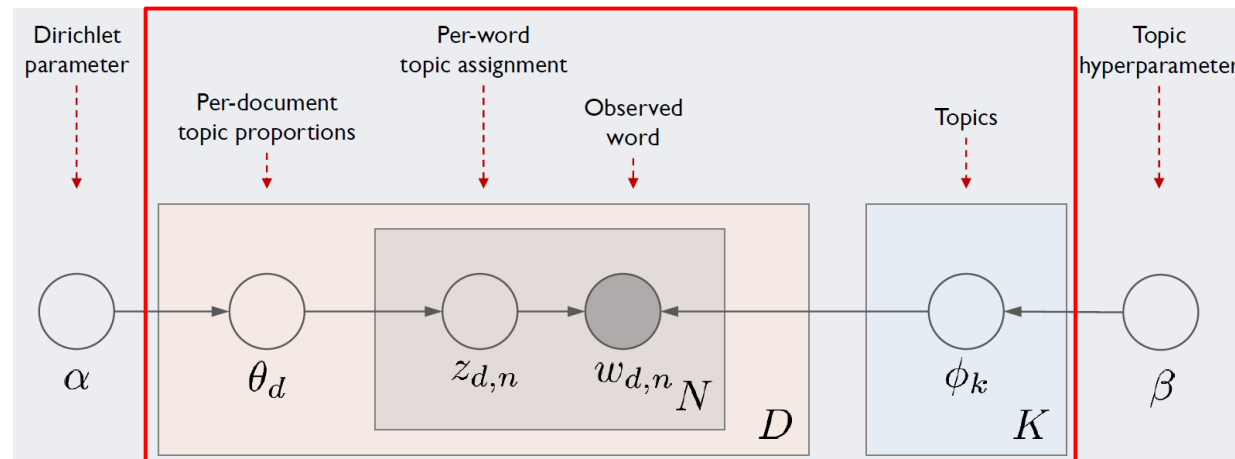
LDA(Latent Dirichlet Allocation)

- 하이퍼 파라미터
 - K : 전체 토픽 수



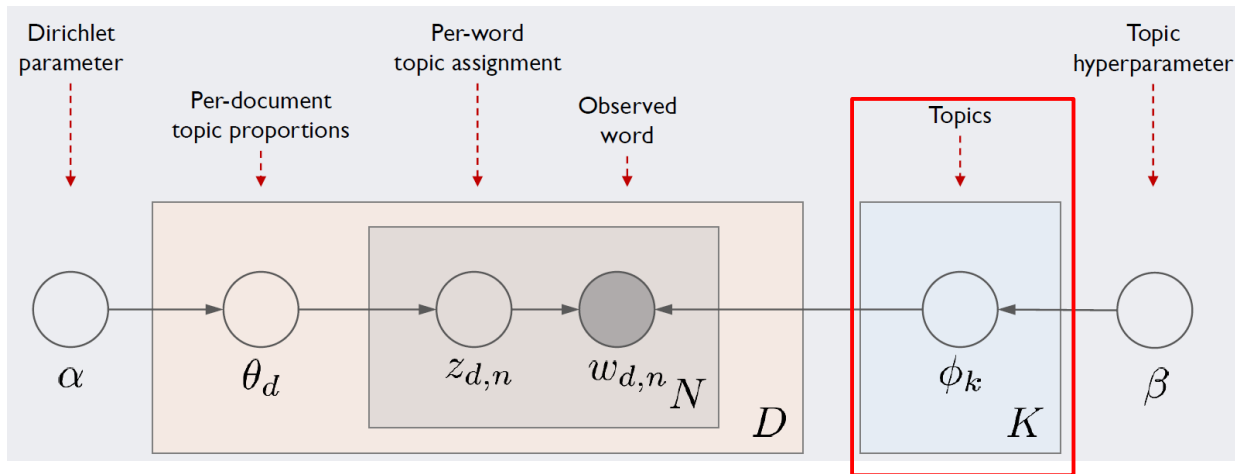
LDA(Latent Dirichlet Allocation)

- 모델 아키텍처
 - **D**: 말뭉치 전체 문서 개수
 - **K**: 전체 토픽 수
 - **N**: **d**번째 문서의 단어 수
 - 하이퍼 파라미터: α, β, K
 - 하이퍼 파라미터는 모델의 성능에 영향을 줄 수 있는 매개변수로 분석가가 정해야 됨



LDA(Latent Dirichlet Allocation)

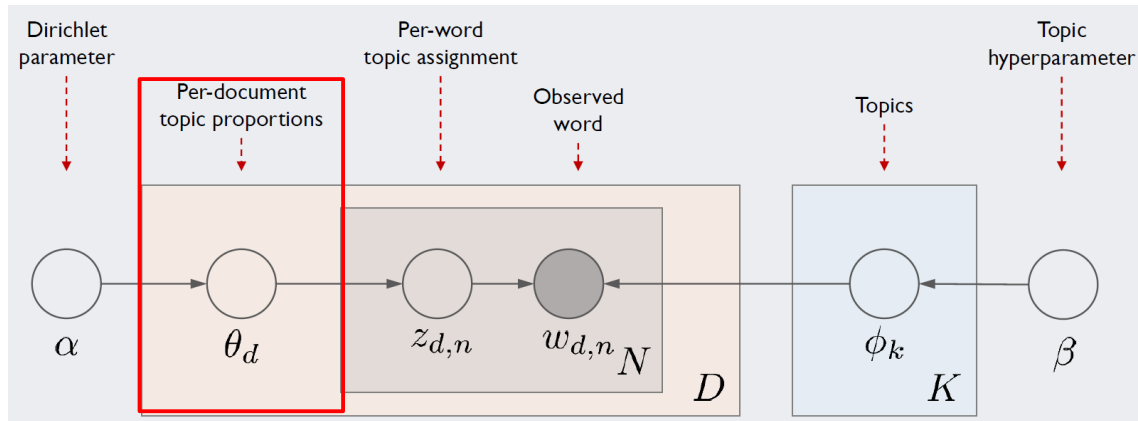
- 모델 아키텍처
 - β 를 인자로 받아 K개 만큼 반복하여 $w_{d,n}$ 으로 보냄
 - ϕ_k : k번째 토픽에 해당하는 벡터
 - 각 열의 합은 1



	ϕ_1	ϕ_2	ϕ_3
Terms	Topic 1	Topic 2	Topic 3
Baseball	0.000	0.000	0.200
Basketball	0.000	0.000	0.267
Boxing	0.000	0.000	0.133
Money	0.231	0.313	0.400
Interest	0.000	0.312	0.000
Rate	0.000	0.312	0.000
Democrat	0.269	0.000	0.000
Republican	0.115	0.000	0.000
Cocus	0.192	0.000	0.000
President	0.192	0.063	0.000

LDA(Latent Dirichlet Allocation)

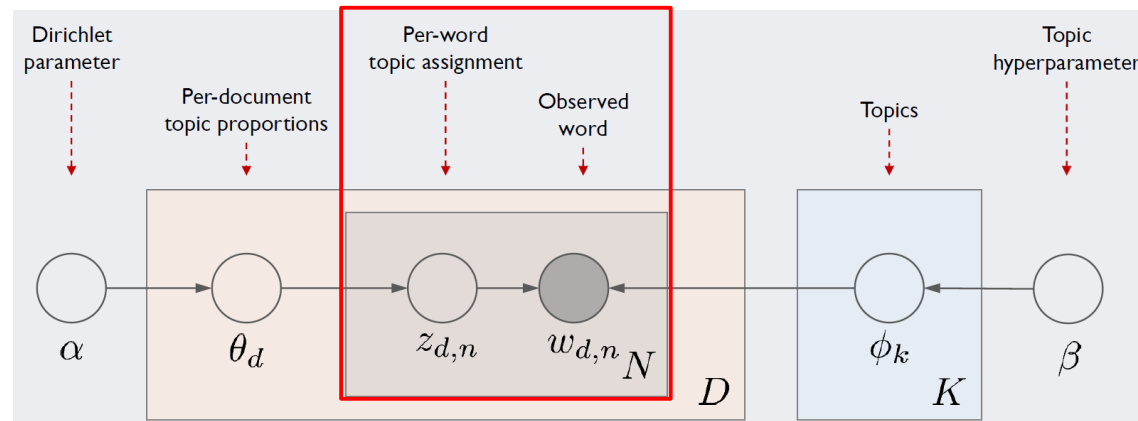
- θ_d : d번째 문서가 가진 토픽 비중을 나타내는 벡터
 - θ_d 는 확률이기 때문에 모든 요소의 합이 1
 - 하이퍼파라미터 α 에 영향을 받음



	Docs	Topic 1	Topic 2	Topic 3
θ_1	Doc 1	0.400	0.000	0.600
	Doc 2	0.000	0.600	0.400
	Doc 3	0.375	0.625	0.000
	Doc 4	0.000	0.375	0.625
θ_5	Doc 5	0.500	0.000	0.500
	Doc 6	0.500	0.500	0.000

LDA(Latent Dirichlet Allocation)

- $z_{d,n}$: d번째 문서 n번째 단어가 어떤 토픽에 해당하는지 할당해주는 역할
- $w_{d,n}$: 문서에 등장하는 단어를 할당해주는 역할
 - 그림과 같이 ϕ_k 와 $z_{d,n}$ 에 같이 영향을 받음



LDA(Latent Dirichlet Allocation)

- 1_cut_file.py
 - 지도학습 시간이 오래 걸려서 전체 데이터의 10%(100,000개)만 사용하여 실습

```
cut_file.py x
1 num = 100000
2 with open('abcnews-date-text.csv', 'r') as fr:
3     data = fr.readlines()[ :num+1]
4 with open('abcnews-date-text_%s.csv' % num, 'w') as fw:
5     fw.writelines(data)
```

TopicModeling

- million-headlines
- _useless_main.py
- abcnews-date-text.csv
- abcnews-date-text_100000.csv
- calc_topic_cnt.py
- calc_topic_cnt2.py
- cut_file.py
- main2.py
- million-headlines.zip

LDA(Latent Dirichlet Allocation)

- 2_main2.py
 - NLTK를 이용하여 LDA 알고리즘 실습
 - NLTK: Natural Language Toolkit
 - Natural Language: 자연어(영어, 한국어, 일본어, 중국어)
 - Toolkit: 도구
 - NLTK 강의 자료 참조
 - 자연어 처리 및 텍스트 분석용 Python Package

main2.py

- `pd.read_csv('abcnews_date-text_100000.csv', error_bad_lines=False)`
 - csv 형태로 데이터를 읽는 함수
 - `error_bad_lines = False`: csv 형식이 아닌 줄은 건너뛰는 옵션
- `data.head(5)`
 - 전체 데이터에서 맨 위에 있는 5개 데이터를 출력함(column 정보도 같이 출력)

```
1 import pandas as pd
2 import nltk
3 from nltk.corpus import stopwords
4
5 data = pd.read_csv('abcnews-date-text_100000.csv', error_bad_lines=False)
6 print(len(data))
7 print(data.head(5))
```

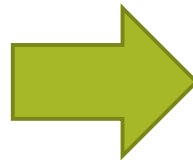
```
100000
      publish_date      headline_text
0      20030219  aba decides against community broadcasting lic...
1      20030219    act fire witnesses must be aware of defamation
2      20030219  a g calls for infrastructure protection summit
3      20030219      air nz staff in aust strike for pay rise
4      20030219    air nz strike to affect australian travellers
```


main2.py

- Topic Modeling을 수행할 때, `publish_date` 컬럼은 필요 없기 때문에 제거함
 - `publish_date`는 출판 날짜이기 때문에 텍스트 마이닝에 관계 없음
 - 열은 사라지고 행 단위로 집계하고 싶은 경우 `axis=1`

```
9 text = data[['headline_text']]
10 print(text.head(5))
11 text2 = text.copy()
12 text2['headline_text'] = text.apply(lambda row: nltk.word_tokenize(row['headline_text']), axis=1)
13 print(text2.head(5))
```

	headline_text
0	aba decides against community broadcasting lic...
1	act fire witnesses must be aware of defamation
2	a g calls for infrastructure protection summit
3	air nz staff in aust strike for pay rise
4	air nz strike to affect australian travellers



	headline_text
0	[aba, decides, against, community, broadcastin...
1	[act, fire, witnesses, must, be, aware, of, de...
2	[a, g, calls, for, infrastructure, protection,...
3	[air, nz, staff, in, aust, strike, for, pay, r...
4	[air, nz, strike, to, affect, australian, trav...

main2.py

- pandas의 apply() 함수
 - 행 또는 열 또는 전체 원소에 대해 특정 연산을 적용시킬 때 사용하는 함수
 - 행은 사라지고 열 단위로 집계하고 싶은 경우 **axis=0**
 - 열은 사라지고 행 단위로 집계하고 싶은 경우 **axis=1**

```
9 text = data[['headline_text']]
10 text.head(5)
11 text2 = text.copy()
12 text2['headline_text'] = text.apply(lambda row: nltk.word_tokenize(row['headline_text']), axis=1)
13 print(text2.head(5))
```

pandas/ex1.py

- pandas의 apply() 함수
 - 행 또는 열 또는 전체 원소에 대해 특정 연산을 적용시킬 때 사용하는 함수

```
ex1.py x
1 import numpy as np
2 import pandas as pd
3
4 a = pd.DataFrame({'국어': [51, 65, 78], \
5                  '수학': [80, 90, 100]}, \
6                  index=['Kim', 'Lee', 'Choi'])
7 print(a)
8 b = a.apply(np.sqrt)
9 print(b)
```

Run: ex1 x

C:\Users\ktw13\AppData\Local\I

	국어	수학
Kim	51	80
Lee	65	90
Choi	78	100

국어 수학

Kim	7.141428	8.944272
Lee	8.062258	9.486833
Choi	8.831761	10.000000

pandas/ex2.py

- pandas의 apply() 함수
 - 행 또는 열 또는 전체 원소에 대해 특정 연산을 적용시킬 때 사용하는 함수
 - 행은 사라지고 열 단위로 집계하고 싶은 경우 **axis=0**
 - 열은 사라지고 행 단위로 집계하고 싶은 경우 **axis=1**

```
ex2.py x
1  import numpy as np
2  import pandas as pd
3
4  a = pd.DataFrame({'국어': [51, 65, 78], \
5                    '수학': [80, 90, 100]}, \
6                    index=['Kim', 'Lee', 'Choi'])
7
8  print(a)
9  b = a.apply(np.average, axis=0)
10 print(b)
11 c = a.apply(np.average, axis=1)
12 print(c)
```

	국어	수학
Kim	51	80
Lee	65	90
Choi	78	100

국어	64.666667
수학	90.000000
dtype: float64	

axis=0

Kim	65.5
Lee	77.5
Choi	89.0
dtype: float64	

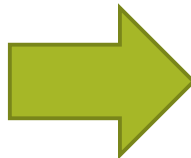
axis=1

main2.py

- 불용어 제거(stopwords)
 - a, in, for와 같은 단어가 제거되는 것을 볼 수 있음
 - 전치사와 같은 단어는 데이터 분석에 거의 도움이 되지 않고 분석하기가 어려움

```
14 stop = stopwords.words('english')
15 text3 = text2.copy()
16 text3['headline_text'] = text2['headline_text'].apply(lambda x: [word for word in x if word not in stop])
17 print(text3.head(5))
```

	headline_text
0	[aba, decides, against, community, broadcastin...
1	[act, fire, witnesses, must, be, aware, of, de...
2	[a, g, calls, for, infrastructure, protection,...
3	[air, nz, staff, in, aust, strike, for, pay, r...
4	[air, nz, strike, to, affect, australian, trav...



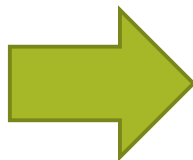
	headline_text
0	[aba, decides, community, broadcasting, licence]
1	[act, fire, witnesses, must, aware, defamation]
2	[g, calls, infrastructure, protection, summit]
3	[air, nz, staff, aust, strike, pay, rise]
4	[air, nz, strike, affect, australian, travellers]

main2.py

- 표제어 추출(lemmatize)
 - broadcasting => broadcast
 - calls => call

```
20 from nltk.stem import WordNetLemmatizer
21 text4 = text3.copy()
22 text4['headline_text'] = text3['headline_text'].apply(lambda x: [WordNetLemmatizer().lemmatize(word, pos='v') for word in x])
23 print(text4.head(5))
```

	headline_text
0	[aba, decides, community, broadcasting, licence]
1	[act, fire, witnesses, must, aware, defamation]
2	[g, calls, infrastructure, protection, summit]
3	[air, nz, staff, aust, strike, pay, rise]
4	[air, nz, strike, affect, australian, travellers]



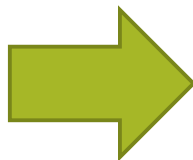
	headline_text
0	[aba, decide, community, broadcast, licence]
1	[act, fire, witness, must, aware, defamation]
2	[g, call, infrastructure, protection, summit]
3	[air, nz, staff, aust, strike, pay, rise]
4	[air, nz, strike, affect, australian, travellers]

main2.py

- 길이가 짧은 단어 제거
 - 길이가 짧은 단어는 유용한 정보가 담겨 있지 않다고 가정하여 제거함
 - Ex) g, nz

```
25 #길이가 짧은 단어 제거
26 tokenized_doc = text4['headline_text'].apply(lambda x: [word for word in x if len(word) > 3])
27 print(tokenized_doc[:5])
```

	headline_text
0	[aba, decide, community, broadcast, licence]
1	[act, fire, witness, must, aware, defamation]
2	[g, call, infrastructure, protection, summit]
3	[air, nz, staff, aust, strike, pay, rise]
4	[air, nz, strike, affect, australian, travellers]



0	[decide, community, broadcast, licence]
1	[fire, witness, must, aware, defamation]
2	[call, infrastructure, protection, summit]
3	[staff, aust, strike, rise]
4	[strike, affect, australian, travellers]

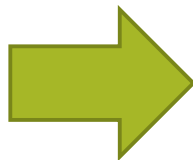
main2.py

- 역토큰화(Detokenize)

- 길이가 짧은 단어를 제거하기 위해 토큰화했던 작업을 다시 되돌리기 위해 역토큰화 작업 수행

```
30 detokenized_doc = []
31 for i in range(len(text4)):
32     t = ' '.join(tokenized_doc[i])
33     detokenized_doc.append(t)
34 text4['headline_text'] = detokenized_doc # 다시 text['headline_text']에 재저장
```

```
0    [decide, community, broadcast, licence]
1    [fire, witness, must, aware, defamation]
2    [call, infrastructure, protection, summit]
3        [staff, aust, strike, rise]
4    [strike, affect, australian, travellers]
```



Name: headline_text, dtype: object

```
0    decide community broadcast licence
1    fire witness must aware defamation
2    call infrastructure protection summit
3        staff aust strike rise
4    strike affect australian travellers
```


main2.py

- TfidfVectorizer
 - 1,000개의 단어를 가지고 행렬을 만들기 위해 TF-IDF 행렬 만들기

```
37 from sklearn.feature_extraction.text import TfidfVectorizer
38 vectorizer = TfidfVectorizer(stop_words='english', max_features=1000) # 상위 1,000개의 단어를 보존
39 X = vectorizer.fit_transform(text4['headline_text'])
40
41 from sklearn.decomposition import LatentDirichletAllocation
42 lda_model = LatentDirichletAllocation(n_components=10, max_iter=1)
43 lda_top = lda_model.fit_transform(X)
44 terms = vectorizer.get_feature_names() # 단어 집합. 1,000개의 단어가 저장됨.
45
46 def get_topics(components, feature_names, n=10):
47     for idx, topic in enumerate(components):
48         print("Topic %d:" % (idx+1), [(feature_names[i], topic[i].round(2)) for i in topic.argsort()[::-n - 1:-1]])
49
50 get_topics(lda_model.components_, terms)
```

main2.py

- LDA 적용
 - n_components: 토픽의 개수
 - max_iter: 반복 횟수

```
37 from sklearn.feature_extraction.text import TfidfVectorizer
38 vectorizer = TfidfVectorizer(stop_words='english', max_features=1000) # 상위 1,000개의 단어를 보존
39 X = vectorizer.fit_transform(text4['headline_text'])
40
41 from sklearn.decomposition import LatentDirichletAllocation
42 lda_model = LatentDirichletAllocation(n_components=10, max_iter=1)
43 lda_top = lda_model.fit_transform(X)
44 terms = vectorizer.get_feature_names() # 단어 집합. 1,000개의 단어가 저장됨.
45
46 def get_topics(components, feature_names, n=10):
47     for idx, topic in enumerate(components):
48         print("Topic %d:" % (idx+1), [(feature_names[i], topic[i].round(2)) for i in topic.argsort()[::-n - 1:-1]])
49
50 get_topics(lda_model.components_, terms)
```

main2.py

- get_topics()
 - enumerate 함수를 이용하여 LDA 수행 결과 출력

```
37 from sklearn.feature_extraction.text import TfidfVectorizer
38 vectorizer = TfidfVectorizer(stop_words='english', max_features=1000) # 상위 1,000개의 단어를 보존
39 X = vectorizer.fit_transform(text4['headline_text'])
40
41 from sklearn.decomposition import LatentDirichletAllocation
42 lda_model = LatentDirichletAllocation(n_components=10, max_iter=1)
43 lda_top = lda_model.fit_transform(X)
44 terms = vectorizer.get_feature_names() # 단어 집합. 1,000개의 단어가 저장됨.
45
46 def get_topics(components, feature_names, n=10):
47     for idx, topic in enumerate(components):
48         print("Topic %d:" % (idx+1), [(feature_names[i], topic[i].round(2)) for i in topic.argsort()[::-n - 1:-1]])
49
50 get_topics(lda_model.components_, terms)
```

```
Topic 1: [('lead', 214.92), ('seek', 172.07), ('plan', 171.75), ('clai
Topic 2: [('charge', 441.5), ('police', 248.42), ('seek', 166.38), ('v
Topic 3: [('plan', 424.79), ('court', 292.06), ('police', 190.35), ('
Topic 4: [('iraq', 207.87), ('make', 194.72), ('minister', 139.31), ('
Topic 5: [('iraq', 179.29), ('welcome', 109.91), ('govt', 100.15), ('
Topic 6: [('miss', 198.88), ('claim', 188.31), ('warn', 179.98), ('ur
Topic 7: [('govt', 361.31), ('kill', 197.64), ('arrest', 186.69), ('cc
Topic 8: [('face', 167.35), ('council', 146.31), ('support', 136.18),
Topic 9: [('urge', 195.58), ('concern', 181.4), ('australia', 152.56),
Topic 10: [('police', 269.05), ('murder', 167.72), ('deal', 147.74), ('
```

main2.py

- get_topics() 분석 - 1
 - get_topics() 함수를 호출하기 전 변수 값 확인
 - lda_model.components_
 - terms

```
46 def get_topics(components, feature_names, n=10):
47     for idx, topic in enumerate(components):
48         print("Topic %d:" % (idx+1), [(feature_na
49
50 1 get_topics(lda_model.components_, terms)
```

The screenshot shows a Jupyter Notebook interface. The top part displays the execution of the `get_topics(lda_model.components_, terms)` function. Below the code, the Variables panel shows the state of the variables. The `lda_model.components_` variable is an ndarray of shape (10, 1000) with dtype float64. The `terms` variable is a list of 10 topics, each represented as a list of words. The output of the function is displayed as a list of 10 topics, each with a list of words.

```
51 get_topics(lda_model.components_, terms)
52
53
54 #n_components: topic 개수
55 #max_iter: The maximum number of iterations
```

Variables

- `lda_model.components_` = (ndarray) [[32.09096241 2.78792493 5.38320531 ... 6.05714111 5.01728644Wn 7.50055209]Wn 1.5797632 5.5970029 ...]
- `min` = (float64) 0.10059404529698444
- `max` = (float64) 366.4361975713806
- `shape` = (tuple) <class 'tuple'>: (10, 1000)
- `dtype` = (dtype) float64
- `size` = (int) 10000
- `array` = (NdArrayItemsContainer) <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.NdArrayItemsContainer object at 0x0000018C563A45C0>
- `terms` = (list) <class 'list'>: ['2004', 'abbott', 'aboriginal', 'abuse', 'acc', 'accept', 'access', 'accident', 'accuse', 'aceh', 'action', 'address', 'adelaide', 'admit', 'affec']
- `0000` = (str) '2004'
- `0001` = (str) 'abbott'
- `0002` = (str) 'aboriginal'
- `0003` = (str) 'abuse'
- `0004` = (str) 'acc'
- `0005` = (str) 'accept'
- `0006` = (str) 'access'
- `0007` = (str) 'accident'
- `0008` = (str) 'accuse'

main2.py

- `get_topics()` 분석 - 2
 - tuple, `argsort()`, for문, list comprehension으로 복잡하게 구성되어 있음
 - idx: 0
 - topic: [32.09096241 ...]

```
def get_topics(components, feature_names, n=10):  
    components: [[32.09096241  2.78792493  5.38320531 ...  6.05714111  
    for idx, topic in enumerate(components):  
        idx: 0  topic: [32.09096241  2.78792493  5.38320531 12.24528302  6.23  
2 print("Topic %d:" % (idx+1), [(feature_names[i], topic[i].round(2)) for i in topic.argsort()[::-1:-1]])
```

```
get_topics(lda_model.components, terms)
```

```
lda_model.components_ = (ndarray) [[32.09096241  2.78792493  5.38320531 ...  6.05714111  5.01728644Wn  7.50055209]Wn [ 1.5797632  5.  
terms = {list} <class 'list'>: ['2004', 'abbott', 'aboriginal', 'abuse', 'acc', 'accept', 'access', 'accident', 'accuse', 'aceh', 'action', 'address', 'adelaide', 'ac  
components = (ndarray) [[32.09096241  2.78792493  5.38320531 ...  6.05714111  5.01728644Wn  7.50055209]Wn [ 1.5797632  5.5970029  1  
feature_names = {list} <class 'list'>: ['2004', 'abbott', 'aboriginal', 'abuse', 'acc', 'accept', 'access', 'accident', 'accuse', 'aceh', 'action', 'address', 'ade  
idx = {int} 0  
n = {int} 10  
topic = (ndarray) [32.09096241  2.78792493  5.38320531 12.24528302  6.23928561  0.76320207Wn  5.44842603 33.86424847 46.92285926
```

main2.py

- get_topics() 분석 - 3
 - 코드를 하나씩 쪼개어 출력 값을 확인하면서 분석하는 것이 공부에 많은 도움이 됨
 - PyCharm의 Debug 기능 활용하기
 - artsort() => <https://stml.tistory.com/12>

```
def get_topics(components, feature_names, n=10):  
    for idx, topic in enumerate(components):  
        topic_list = topic.argsort()  
        #print(topic_list)  
        topic_list2 = topic_list[:-(n+1):-1]  
        print("\nTopic %d:" % (idx+1), end=' ')  
        for i in topic_list2:  
            print_tuple = (feature_names[i], topic[i].round(2))  
            print(print_tuple, end='')
```

```
Topic 1: ('iraq', 264.67)('probe', 233.91)('police', 206.32)(  
Topic 2: ('police', 304.9)('plan', 250.17)('urge', 176.39)('h  
Topic 3: ('report', 239.73)('govt', 168.81)('police', 168.76)  
Topic 4: ('continue', 294.8)('boost', 205.41)('fund', 195.24)  
Topic 5: ('kill', 221.4)('attack', 180.97)('govt', 166.6)('ap  
Topic 6: ('govt', 210.01)('police', 198.62)('charge', 192.63)  
Topic 7: ('plan', 207.24)('govt', 166.54)('concern', 166.48)(  
Topic 8: ('charge', 253.88)('court', 231.5)('plan', 167.68)('h  
Topic 9: ('consider', 207.57)('claim', 170.9)('plan', 144.07)  
Topic 10: ('play', 133.8)('police', 118.92)('kill', 114.38)('h
```

LDA 토픽 개수 지정

- 토픽 모델링의 토픽 개수(K)를 지정하는 방법
 - 분석을 하고자 하는 문서에 대해 자세히 알고 문서의 개수가 적은 경우, 적절한 토픽 개수를 정할 수 있음
 - 하지만 대부분의 연구에서 적절한 토픽 개수를 정하기 어렵기 때문에, 통계적인 방법으로 최적의 토픽 개수를 구해야 함
- 통계적 지표
 - Perplexity
 - Topic coherence

LDA 토픽 개수 지정

- Perplexity
 - 혼란도(사전적 의미: 당혹, 혼란, 곤혹)
 - 토픽 개수를 늘릴수록 **perplexity**는 감소하는 경향이 보이며, 특정 토픽 개수 지점을 지나면 더 이상 **perplexity**는 감소하지 않고 수렴하는 지점이 나타남
 - 수렴하는 지점 => 최종 **perplexity**
 - **Perplexity**가 작으면 작을수록 토픽 모델이 실제 문서 내용을 잘 분류한다는 뜻으로 해석할 수 있음

LDA 토픽 개수 지정

- Topic Coherence

- coherence => 일관성
- 실제로 사람이 해석하기에 적합한 평가 척도를 만들기 위해 제시된 척도
- 토픽 모델링 결과로 나온 주제들에 대해 각각의 주제에서 상위 N개의 단어 추출
- 모델링이 잘 될수록 한 주제 안에는 의미론적으로 유사한 단어가 많이 모여있음
- 따라서 상위 단어 간의 유사도를 계산하여 평균을 구하면 실제로 해당 주제가 의미론적으로 일치하는 단어들끼리 모여있는지 파악할 수 있음

LDA 토픽 개수 지정

- `calc_topic_cnt1.py`
 - `RegexTokenizer('[\w]+')`: a-z, A-Z, 0-9, _을 찾는 정규식
 - `PorterStemmer()`: 단어의 접미사를 제거하는 라이브러리
 - Ex) cats => cat, conflated => conflate, troubling => trouble, relational => relate

```
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from gensim import corpora
import gensim
from nltk.tokenize import RegexTokenizer

tokenizer = RegexTokenizer('[\w]+')
stop_words = stopwords.words('english')
p_stemmer = PorterStemmer()
```

LDA 토픽 개수 지정

- calc_topic_cnt1.py
 - 빈 리스트 doc_set에 10개의 문장 저장

```
doc_a = "Broccoli is good to eat. My brother likes to eat good broccoli, but not my mother."  
doc_b = "My mother spends a lot of time driving my brother around to baseball practice."  
doc_c = "Some health experts suggest that driving may cause increased tension and blood pressure."  
doc_d = "I often feel pressure to perform well at school, but my mother never seems to drive my bro  
doc_e = "Health professionals say that broccoli is good for your health."  
doc_f = "Big data is a term used to refer to data sets that are too large or complex for traditiona  
doc_g = "Data with many cases offer greater statistical power, while data with higher complexity ma  
doc_h = "Big data was originally associated with three key concepts: volume, variety, and velocity.  
doc_i = "A 2016 definition states that 'Big data represents the information assets characterized by  
doc_j = "Data must be processed with advanced tools to reveal meaningful information."  
  
doc_set = [doc_a, doc_b, doc_c, doc_d, doc_e, doc_f, doc_g, doc_h, doc_i, doc_j]
```

LDA 토픽 개수 지정

- `calc_topic_cnt1.py`
 - `lower()`: 소문자 변환 함수
 - `tokenize`, `stopwords`, `stemming`을 통해 데이터 전처리 수행

```
texts = []  
  
for w in doc_set:  
    raw = w.lower()  
    tokens = tokenizer.tokenize(raw)  
    stopped_tokens = [i for i in tokens if not i in stop_words]  
    stemmed_tokens = [p_stemmer.stem(i) for i in stopped_tokens]  
    texts.append(stemmed_tokens)
```

LDA 토픽 개수 지정

- calc_topic_cnt1.py
 - texts 리스트들을 dictionary 형태로 변환

```
calc_topic_cnt x
C:\Users\ktw13\AppData\Local\Programs\Python\Python37\python.exe C:/Users/ktw13/Desktop/data_analysis/3.Crawling_TextMining/Topic
[['brocolli', 'good', 'eat', 'brother', 'like', 'eat', 'good', 'brocolli', 'mother'], ['mother', 'spend', 'lot', 'time', 'drive',
Dictionary(85 unique tokens: ['brocolli', 'brother', 'eat', 'good', 'like']...)]
[[ (0, 2), (1, 1), (2, 2), (3, 2), (4, 1), (5, 1)], [(1, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1)], [
[(0, '0.030*"good" + 0.028*"mother" + 0.027*"brocolli" + 0.025*"brother" + 0.023*"eat"), (1, '0.071*"data" + 0.040*"big" + 0.026
[(0, 0.042102076), (1, 0.033707727), (2, 0.9241902)]
```

```
print(texts)
dictionary = corpora.Dictionary(texts)
print(dictionary)
corpus = [dictionary.doc2bow(text) for text in texts]
print(corpus)
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=3,
                                             id2word=dictionary)

print(ldamodel.print_topics(num_words=5))
print(ldamodel.get_document_topics(corpus)[0])
```

LDA 토픽 개수 지정

- calc_topic_cnt1.py
 - doc2bow(): (단어, 빈도수) 형태로 리스트로 변환하는 함수

```
print(texts)
dictionary = corpora.Dictionary(texts)
print(dictionary)
corpus = [dictionary.doc2bow(text) for text in texts]
print(corpus)
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=3,
                                             id2word=dictionary)

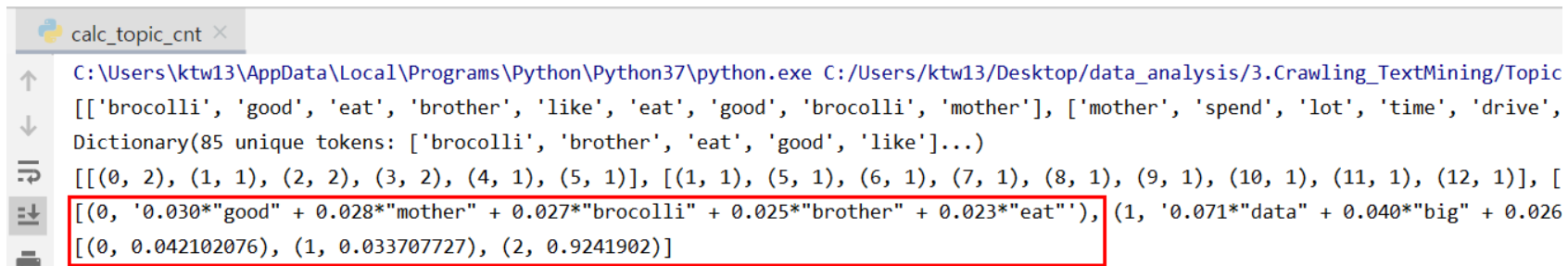
print(ldamodel.print_topics(num_words=5))
print(ldamodel.get_document_topics(corpus)[0])
```

```
calc_topic_cnt x
C:\Users\ktw13\AppData\Local\Programs\Python\Python37\python.exe C:/Users/ktw13/Desktop/data_analysis/3.Crawling_TextMining/Topic
[['broccoli', 'good', 'eat', 'brother', 'like', 'eat', 'good', 'broccoli', 'mother'], ['mother', 'spend', 'lot', 'time', 'drive',
Dictionary(85 unique tokens: ['broccoli', 'brother', 'eat', 'good', 'like']...)
[[ (0, 2), (1, 1), (2, 2), (3, 2), (4, 1), (5, 1)], [(1, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1)], [
[(0, '0.030*"good" + 0.028*"mother" + 0.027*"broccoli" + 0.025*"brother" + 0.023*"eat"), (1, '0.071*"data" + 0.040*"big" + 0.026
[(0, 0.042102076), (1, 0.033707727), (2, 0.9241902)]
```

LDA 토픽 개수 지정

- calc_topic_cnt1.py
 - LdaModel을 통해 토픽 모델링 수행

```
print(texts)
dictionary = corpora.Dictionary(texts)
print(dictionary)
corpus = [dictionary.doc2bow(text) for text in texts]
print(corpus)
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=3,
                                             id2word=dictionary)
print(ldamodel.print_topics(num_words=5))
print(ldamodel.get_document_topics(corpus)[0])
```

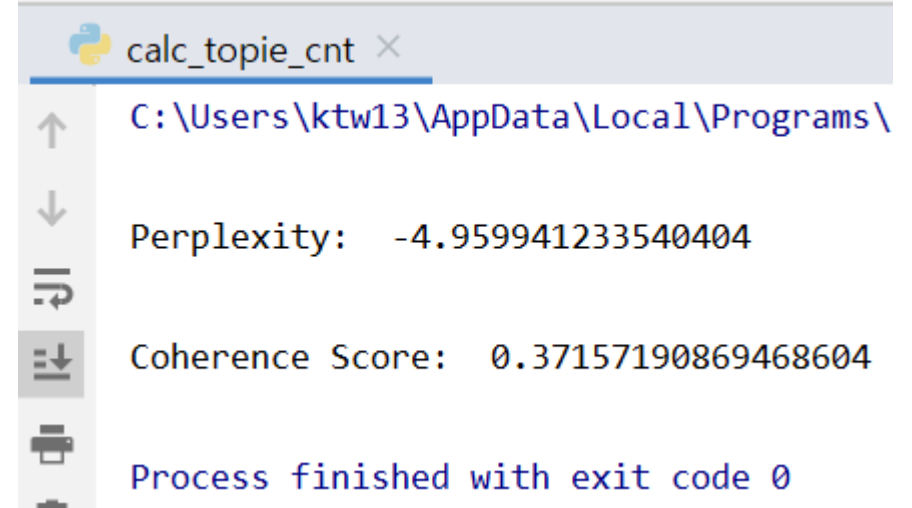


```
calc_topic_cnt x
C:\Users\ktw13\AppData\Local\Programs\Python\Python37\python.exe C:/Users/ktw13/Desktop/data_analysis/3.Crawling_TextMining/Topic
[['broccoli', 'good', 'eat', 'brother', 'like', 'eat', 'good', 'broccoli', 'mother'], ['mother', 'spend', 'lot', 'time', 'drive',
Dictionary(85 unique tokens: ['broccoli', 'brother', 'eat', 'good', 'like']...)
[[ (0, 2), (1, 1), (2, 2), (3, 2), (4, 1), (5, 1)], [(1, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1)], [
[(0, '0.030*\"good\" + 0.028*\"mother\" + 0.027*\"broccoli\" + 0.025*\"brother\" + 0.023*\"eat\"'), (1, '0.071*\"data\" + 0.040*\"big\" + 0.026
[(0, 0.042102076), (1, 0.033707727), (2, 0.9241902)]
```

LDA 토픽 개수 지정

- calc_topic_cnt2.py
 - Topic coherence를 구하기 위한 라이브러리를 불러온 후 perplexity와 topic coherence 계산
 - CoherenceModel
 - topn 파라미터는 상위 N개의 단어를 이용하여 유사도를 계산하라는 의미

```
from gensim.models import CoherenceModel
print('\nPerplexity: ', ldamodel.log_perplexity(corpus))
coherence_model_lda = CoherenceModel(model=ldamodel, texts=texts,
                                     dictionary=dictionary, topn=10)
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```



```
calc_topic_cnt x
C:\Users\ktw13\AppData\Local\Programs\
Perplexity: -4.959941233540404
Coherence Score: 0.37157190869468604
Process finished with exit code 0
```

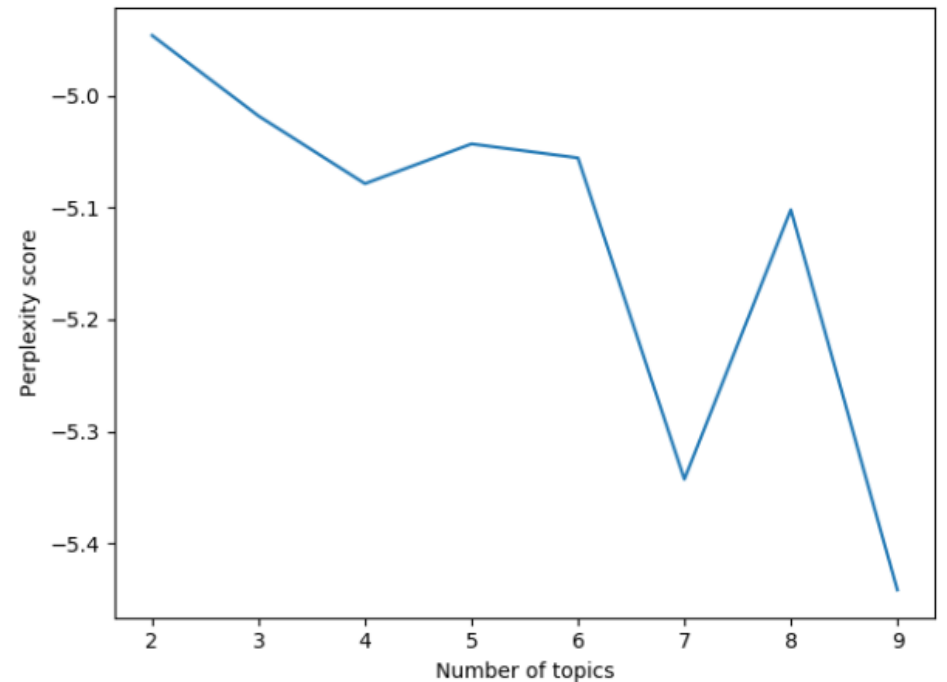

LDA 토픽 개수 지정

- `calc_topic_cnt2.py`
 - 토픽의 개수를 달리하여 **Perplexity score** 계산
 - Topic 수가 4, 7, 9 일 때 낮음

```
import matplotlib.pyplot as plt
perplexity_values = []

for i in range(2, 10):
    ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=i, id2word=dictionary)
    perplexity_values.append(ldamodel.log_perplexity(corpus))

x = range(2, 10)
plt.plot(x, perplexity_values)
plt.xlabel("Number of topics")
plt.ylabel("Perplexity score")
plt.show()
```



LDA 토픽 개수 지정

- `calc_topic_cnt2.py`
 - 토픽의 개수를 달리하여 **Coherence score** 계산
 - 4, 6, 8 지점에서 높은 수치를 보임

```
coherence_values = []
for i in range(2, 10):
    ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=i,
                                                id2word=dictionary)
    coherence_model_lda = CoherenceModel(model=ldamodel, texts=texts,
                                         dictionary=dictionary, topn=10)
    coherence_lda = coherence_model_lda.get_coherence()
    coherence_values.append(coherence_lda)
x = range(2, 10)
plt.plot(x, coherence_values)
plt.xlabel("Number of topics")
plt.ylabel("coherence score")
plt.show()
```

