딥 러닝을 이용한 자연어 처리 입문

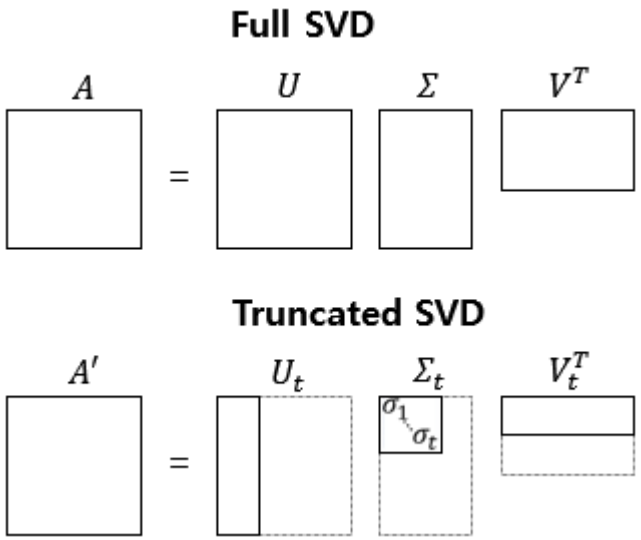
19장 토픽 모델링

발표자 : 김성윤

목차

1. 잠재 의미 분석 (LSA) – 특징

- BoW 기반 DTM과 TF-IDF는 빈도 수를 이용하여 단어의 의미를 고려하지 못함
- 이를 위한 대안이 LSA 방법
- Full *SVD에서 나온 3개의 행렬에서 일부 벡터들을 삭제시킨 절단된(Truncated) SVD 사용
- 절단된 SVD는 상위값 t개 혹은 t열까지만 남게 됨 (t는 토픽 수를 반영한 하이퍼파라미터)
- 절단된 SVD에서 A를 복구할 수 없음
- t가 클 경우 다양한 의미를, t가 작을 경우 노이즈 제거(설명력이 높은 정보만 남김)의 효과를 가짐.



*SVD : 선형대수학의 특이값 분해

$$A = U\Sigma V^{\mathrm{T}}$$

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

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

$\Sigma : m \times n$ 직사각 대각행렬

1. 잠재 의미 분석 (LSA) – SVD 특징

- (U 행렬의 크기) = (문서의 개수) x (토픽의 수, t의 값)
- U 행렬의 각 **행**은 잠재 의미를 표현하기 위한 수치화 된 각각의 **문서 벡터**

- (V^T행렬의 크기) = (토픽의 수, t의 값) x (단어의 개수의 크기)
- V^T 행렬의 각 **열**은 잠재 의미를 표현하기 위한 수치화된 각각의 **단어 벡터**

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

Full SVD 코드

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

```
[[0 0 0 1 0 1 1 0 0]
 [0 0 0 1 1 0 1 0 0]
 [0 1 1 0 2 0 0 0 0]
 [1 0 0 0 0 0 0 1 1]]
[[ 0.   -0.17 -0.17  1.08  0.12  0.62  1.08 -0.   -0.  ]
 [ 0.    0.2   0.2   0.91  0.86  0.45  0.91  0.    0.  ]
 [ 0.    0.93  0.93  0.03  2.05 -0.17  0.03  0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.  ]]
```

Full SVD vs Truncated SVD

0은 0에 가깝게, 1은 1에 가깝게

1. 잠재 의미 분석 (LSA) – 사이킷런 과정

1. 데이터 확인
2. 텍스트 전처리
3. TF-IDF 행렬 만들기 (전처리를 위해 토큰화한 데이터를 역토큰화를 한 후 입력)
4. 토픽 모델링

```
# 역토큰화 (토큰화 작업을 역으로 되돌림)
detokenized_doc = []
for i in range(len(news_df)):
    t = ' '.join(tokenized_doc[i])
    detokenized_doc.append(t)

news_df['clean_doc'] = detokenized_doc
```

역토큰화

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

X = vectorizer.fit_transform(news_df['clean_doc'])
```

TF-IDF 행렬 만들기

```
svd_model = TruncatedSVD(n_components=20, algorithm='randomized', n_iter=100, random_state=122)
svd_model.fit(X)
len(svd_model.components_)
```

토픽 모델링

1. 잠재 의미 분석 (LSA) – 장·단점

1. 장점
- 쉽고 빠르게 구현 가능
- 단어의 잠재적인 의미를 이끌어 낼 수 있음 (문서의 유사도 계산 등에서 좋은 성능을 보임)

2. 단점
- SVD의 특성으로 인해 새로운 데이터를 추가하고 계산 시, 처음부터 다시 계산해야 함

2. 잠재 디리클레 할당 (LDA) – 특징

- LSA의 단점을 개선한 알고리즘
- 가정 :
  1. 문서들은 토픽들의 혼합으로 구성
  2. 토픽들은 확률 분포에 기반하여 단어들을 생성

  예) '나는 이 문서를 작성하기 위해서 이런 주제들을 넣을거고, 이런 주제들을 위해서는 이런 단어들을 넣을 거야.'
- 문서의 생성 과정을 역추적
- 단어의 순서는 신경쓰지 않음 (∵ BoW의 행렬 DTM or TF-IDF 행렬을 입력으로 함)

2. 잠재 디리클레 할당 (LDA) – 특징

- 문서 생성 과정을 가정

1) 문서에 사용할 **단어의 개수 N**을 정합니다.

- Ex) 5개의 단어를 정하였습니다.

<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%**

문서1 : 저는 사과랑 바나나를 먹어요
문서2 : 우리는 귀여운 강아지가 좋아요
문서3 : 저의 깜찍하고 귀여운 강아지가 바나나를 먹어요

<확률 분포 예시>

2) 문서에 사용할 토픽의 혼합을 **확률 분포에 기반**하여 결정합니다.

- Ex) 위 예제와 같이 토픽이 2개라고 하였을 때 강아지 토픽을 60%, 과일 토픽을 40%와 같이 선택할 수 있습니다.

3) 문서에 **사용할 각 단어**를 (아래와 같이) 정합니다.

3-1) 토픽 분포에서 **토픽 T를 확률적으로** 고릅니다.

- Ex) 60% 확률로 강아지 토픽을 선택하고, 40% 확률로 과일 토픽을 선택할 수 있습니다.

3-2) 선택한 토픽 T에서 **단어의 출현 확률 분포에 기반**해 문서에 사용할 단어를 고릅니다.

- Ex) 강아지 토픽을 선택하였다면, 33% 확률로 강아지란 단어를 선택할 수 있습니다. 이제 3)을 반복하면서 문서를 완성합니다.

2. 잠재 디리클레 할당 (LDA) – 알고리즘

1) 사용자가 토픽의 개수 k를 전달

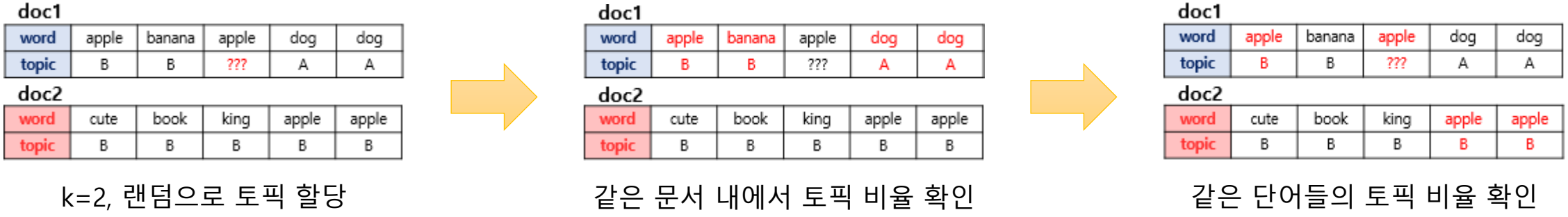
2) 모든 단어에 k개 중 랜덤으로 토픽 할당

3) 모든 문서의 모든 단어를 아래의 두 가지 기준에 따라 토픽 재할당 (iterative)

가정) 임의의 단어 w는 자신은 잘못된 토픽에 할당되어져 있지만, 다른 단어들은 전부 올바른 토픽에 할당되어져 있는 상태라고 가정

- **토픽 재할당의 두 가지 기준**

  1. p(topic t | document d) : 문서 d의 단어들 중 토픽 t에 해당하는 단어들의 비율
  2. p(word w | topic t) : 각 토픽들 t에서 해당 단어 w의 분포



k=2, 랜덤으로 토픽 할당 | 같은 문서 내에서 토픽 비율 확인 | 같은 단어들의 토픽 비율 확인

2. 잠재 디리클레 할당 (LDA) – LSA와 비교

- LSA : DTM을 차원 축소하여 축소 차원에서 근접 단어들을 토픽으로 묶음

- LDA : (단어가 특정 토픽에 존재할 확률)과 (문서에 특정 토픽이 존재할 확률)을 결합확률로 추정하여 토픽을 추출

2. 잠재 디리클레 할당 (LDA) – gensim

```
import gensim
NUM_TOPICS = 20 # 20개의 토픽, k=20
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics = NUM_TOPICS, id2word=dictionary, passes=15)
topics = ldamodel.print_topics(num_words=4)
for topic in topics:
    print(topic)
```

```
(0, '0.015*"drive" + 0.014*"thanks" + 0.012*"card" + 0.012*"system"')
(1, '0.009*"back" + 0.009*"like" + 0.009*"time" + 0.008*"went"')
(2, '0.012*"colorado" + 0.010*"david" + 0.006*"decenso" + 0.005*"tyre"')
(3, '0.020*"number" + 0.018*"wire" + 0.013*"bits" + 0.013*"filename"')
(4, '0.038*"space" + 0.013*"nasa" + 0.011*"research" + 0.010*"medical"')
(5, '0.014*"price" + 0.010*"sale" + 0.009*"good" + 0.008*"shipping"')
(6, '0.012*"available" + 0.009*"file" + 0.009*"information" + 0.008*"version"')
(7, '0.021*"would" + 0.013*"think" + 0.012*"people" + 0.011*"like"')
(8, '0.035*"window" + 0.021*"display" + 0.017*"widget" + 0.013*"application"')
(9, '0.012*"people" + 0.010*"jesus" + 0.007*"armenian" + 0.007*"israel"')
(10, '0.008*"government" + 0.007*"system" + 0.006*"public" + 0.006*"encryption"')
(11, '0.013*"germany" + 0.008*"sweden" + 0.008*"switzerland" + 0.007*"gaza"')
(12, '0.020*"game" + 0.018*"team" + 0.015*"games" + 0.013*"play"')
(13, '0.024*"apple" + 0.014*"water" + 0.013*"ground" + 0.011*"cable"')
(14, '0.011*"evidence" + 0.010*"believe" + 0.010*"truth" + 0.010*"church"')
(15, '0.016*"president" + 0.010*"states" + 0.007*"united" + 0.007*"year"')
(16, '0.047*"file" + 0.035*"output" + 0.033*"entry" + 0.021*"program"')
(17, '0.008*"dept" + 0.008*"devils" + 0.007*"caps" + 0.007*"john"')
(18, '0.011*"year" + 0.009*"last" + 0.007*"first" + 0.006*"runs"')
(19, '0.013*"outlets" + 0.013*"norton" + 0.012*"quantum" + 0.008*"neck"')
```

gensim을 통한 학습 및 단어의 토픽 기여도 확인

```
from gensim import corpora
dictionary = corpora.Dictionary(tokenized_doc)
corpus = [dictionary.doc2bow(text) for text in tokenized_doc]
print(corpus[1]) # 수행된 결과에서 두번째 뉴스 출력. 첫번째 문서의 인덱스는 0
```

```
[(52, 1), (55, 1), (56, 1), (57, 1), (58, 1), (59, 1), (60, 1), (61, 1), (62, 1), (63, 1), (64, 1), (65, 1), (66, 2),
(67, 1), (68, 1), (69, 1), (70, 1), (71, 2), (72, 1), (73, 1), (74, 1), (75, 1), (76, 1), (77, 1), (78, 2), (79, 1),
(80, 1), (81, 1), (82, 1), (83, 1), (84, 1), (85, 2), (86, 1), (87, 1), (88, 1), (89, 1)]
```

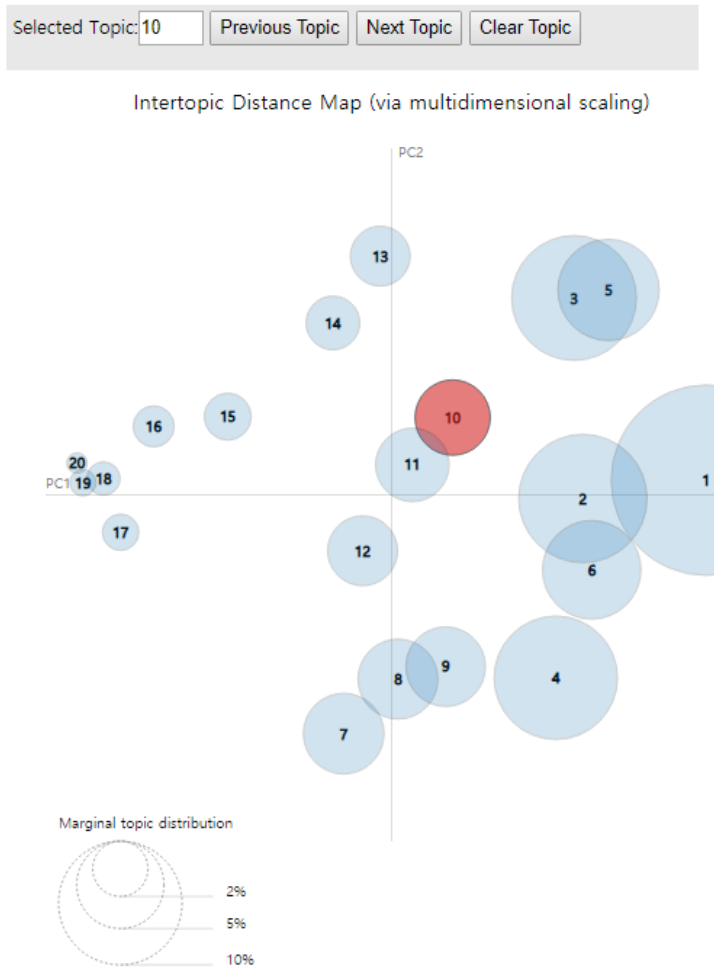
정수 인코딩와 단어 집합 만들기

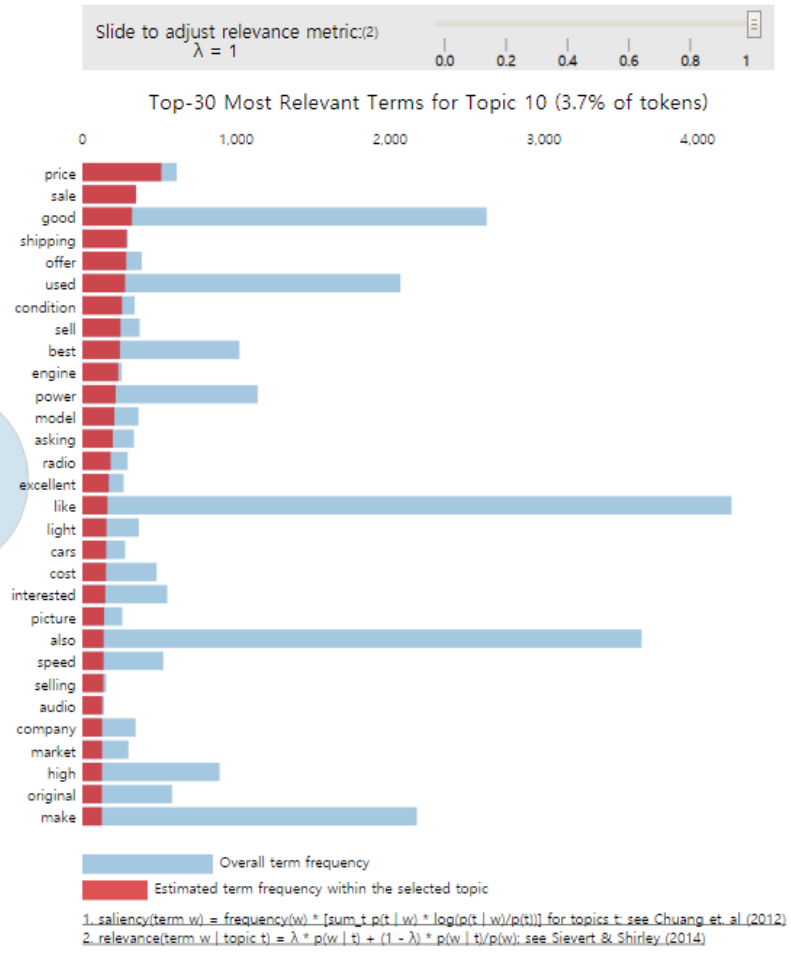
3. 추가적인 tool – pyLDAvis

- LDA 시각화 도구
- 토픽 간 유사도 확인 가능
- 토픽과 관련된 단어 확인 가능

```
import pyLDAvis.gensim_models

pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim_models.prepare(ldamodel, corpus, dictionary)
pyLDAvis.display(vis)
```





3. 추가적인 tool – 복합 토픽 모델 (CTM)

- 문맥을 반영한 토픽 모델(Contextualized Topic Models) 중 하나
- **Contextualized Topic Models** : (문맥을 반영한 BERT의 문서 임베딩의 표현력)과 (기존 토픽 모델의 비지도 학습 능력)을 결합하여 문서에서 주제를 가져오는 토픽 모델

```
tp = TopicModelDataPreparation("paraphrase-distilroberta-base-v1")
training_dataset = tp.fit(text_for_contextual=unpreprocessed_corpus, text_for_bow=preprocessed_documents)
```

문맥을 반영한 문서 임베딩 얻기

```
ctm = CombinedTM(bow_size=len(tp.vocab), contextual_size=768, n_components=50, num_epochs=20)
ctm.fit(training_dataset)
```

토픽 모델 학습

3. 추가적인 tool – BERTopic

- BERT embeddings와 클래스 기반 TF-IDF를 활용
- 주제 설명에서 중요한 단어를 유지하면서 쉽게 해석할 수 있는 조밀한 클러스터를 만듦
- 알고리즘
  1. 텍스트 데이터를 SBERT로 임베딩

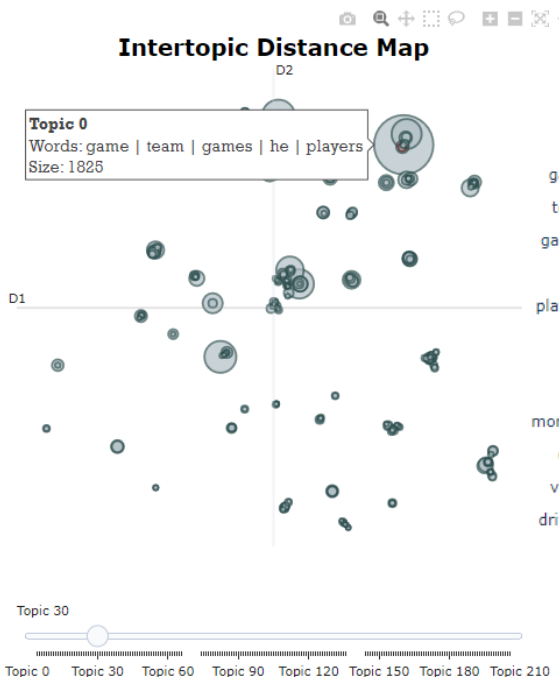
     "paraphrase-MiniLM-L6-v2" : 영어 데이터로 학습된 SBERT

     "paraphrase-multilingual-MiniLM-L12-v2" : 50개 이상의 언어로 학습된 다국어 SBERT
  2. 문서를 군집화 (UMAP, HDBSCAN을 사용하여 차원 축소 및 클러스터링)
  3. 토픽 표현을 생성 (클래스 기반 TF-IDF로 토픽 추출)

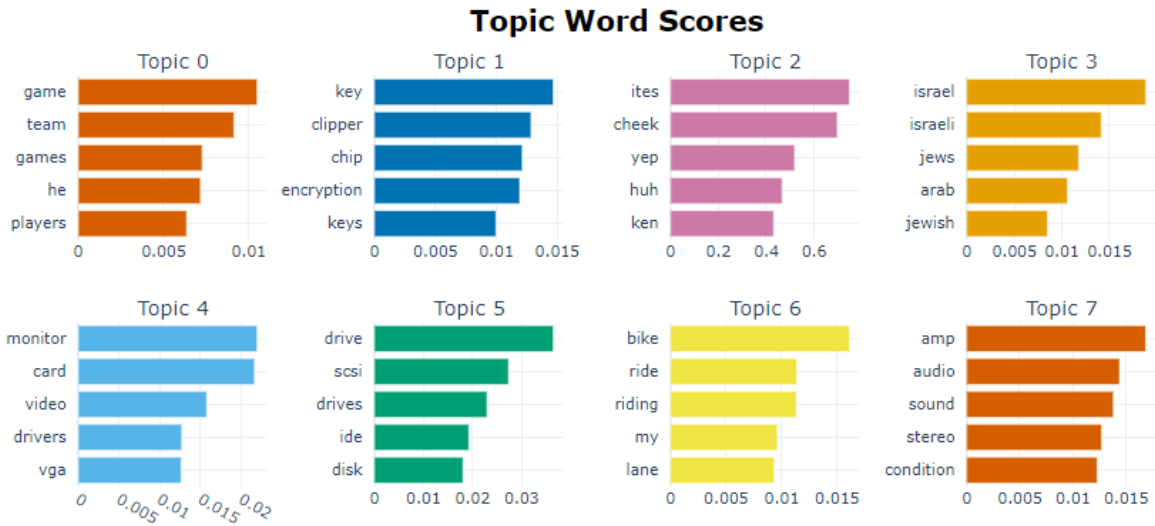
3. 추가적인 tool – BERTopic

```
model = BERTopic()
topics, probabilities = model.fit_transform(docs)
```

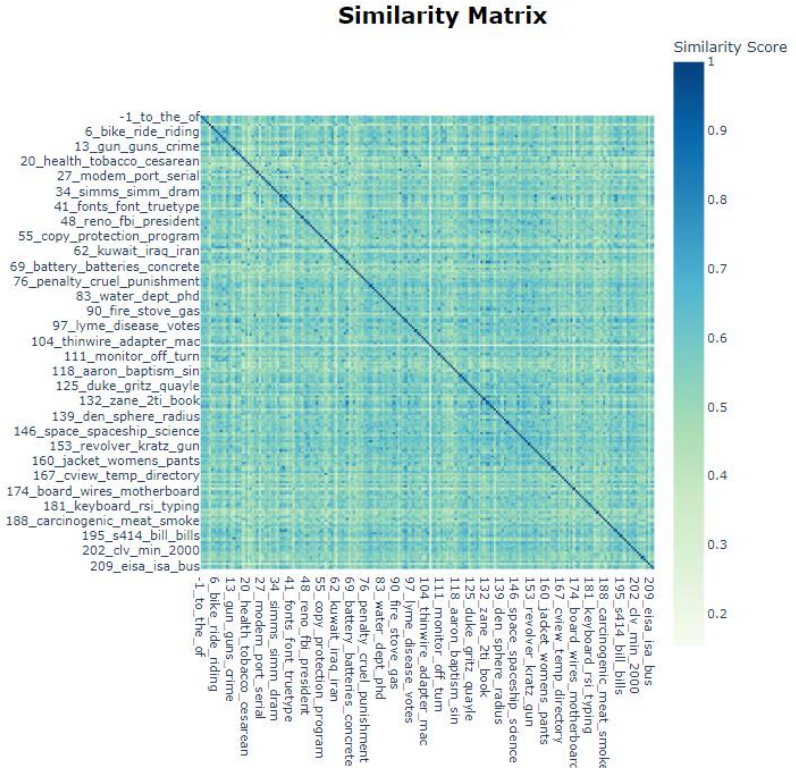
모델 생성



토픽 시각화



단어 시각화



토픽 유사도 시각화