



텍스트 데이터 전처리로 시작하는
Natural **L**anguage **P**rocessing
@싸이그램즈 2018

joeunpark@gmail.com



시작하기 전에 잠시 동화책을 보겠습니다.

오늘도 선생님은 내 쓰기 공책에
빨간색 표시를 하고 소리를 질렀어요.
정말 되어쓰기 따위는
모두 없어야 비었으면 좋겠어요!
되어쓰기는 진짜 진짜 어려워요.
꼭 글자를 뜯어 써야 하나요?

조방귀 금속파매



엄마가 죽을 만들었다.

엄마가 방에 들어가신다.

아빠가 죽을 드신다.

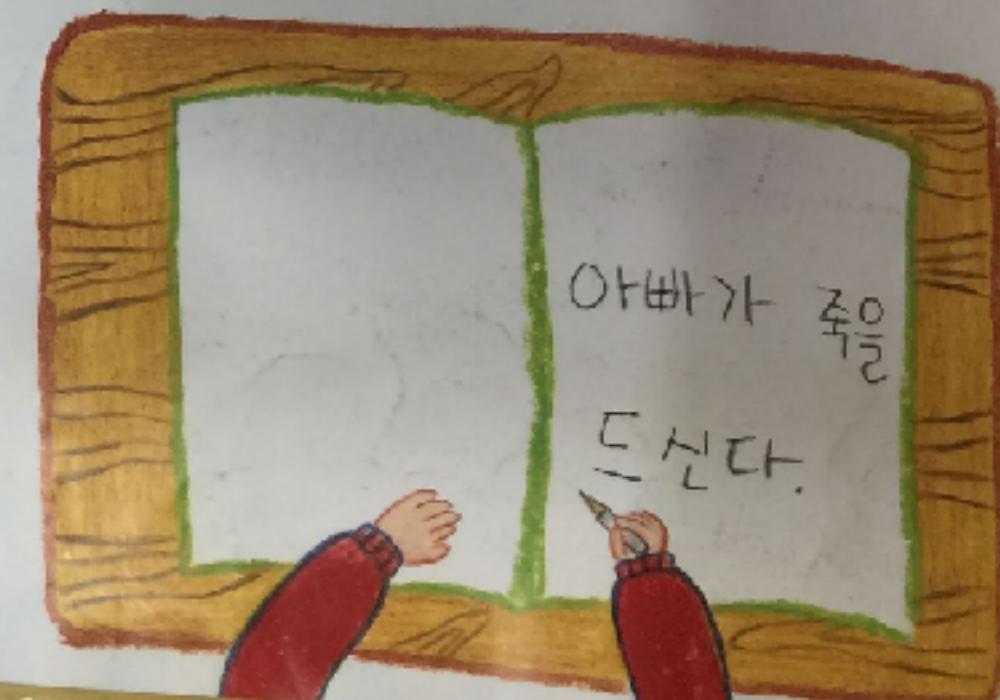
아빠가 빨리 나았으면

좋겠다.



아빠가 가족을 우울하게 씹으며 소리를 질렀어요.
“들려서! 빨리 최대로 안 뛰어 써!”
아마 “틀렸어! 빨리 제대로 안 뛰어 써!”인가 봐요.

나는 아빠에게 살짝 미안한
마음이 들었어요.
나는 잠깐 고민을 하다가
다시 쓰기 공책에 적었어요.
‘아빠가 죽을 드신다.’



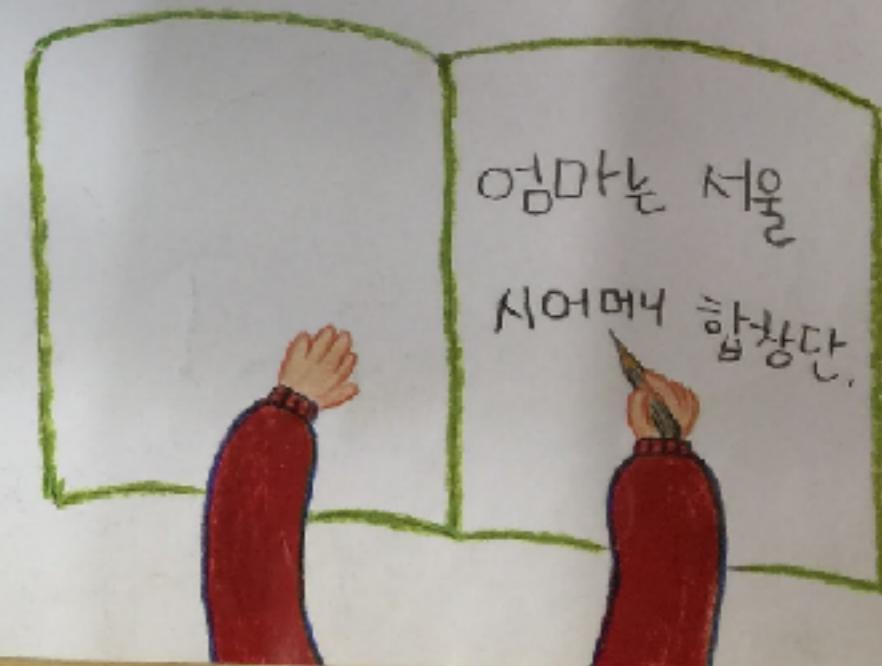


“유, 블씨 합창단
연습 시간이 다 끝네.
오늘은 여기까지 하자.
내일 또 이렇게
이상하게 쓰기단 해봐!”
내 쓰기 공책을 떠나니 가마여
엄마가 말했어요.



“나 더 쓸 거야.”
나는 엄마 손에서 쓰기 공책을 떼앗아
새로운 말을 썼어요.

‘엄마는 서울 시어머니 합창단.’



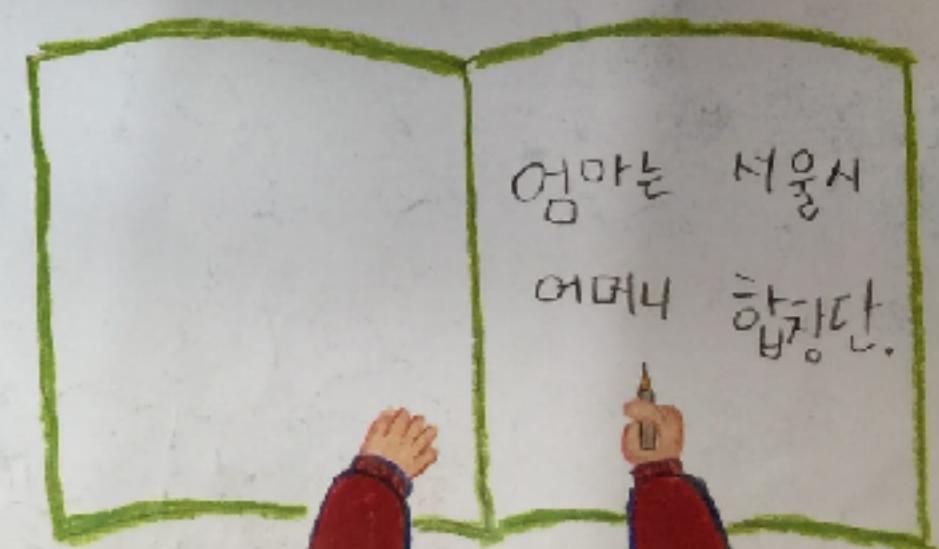


시어머니 = 할머니?



엄마가 갑자기 할머니가 되었어요.
“恚겼어! 내가 왜 빨찌 시어머니야?”
엄마가 엉엉 울었어요.
“아, 이게 아는데, 시…… 어머니?”

엄마의 흰머리와 자글자글한 주름을 보니까
나도 울퍼졌어요. 난 타시 쓰기 공책에 썩었어요.
‘엄마는 서울시 어머니 합장단.’



선생님은 이상하다.



선생님은 이 상하다.



띄어쓰기나 맞춤법에 따라 다른 의미
또, 중의적 표현이나 반어법 등의 표현으로 의미를 찾기 어려움

그래서, 텍스트 전처리가 필요하고
이를 통해 의미를 얻습니다.

그럼, 자연어처리를 **시작하는 분들이**
텍스트 데이터로부터 의미있는 결과를 얻기 위해
어떤 일들을 해볼 수 있을지 간단히 소개해 봅니다.

텍스트 데이터 정제 및 전처리

데이터 정제 및 전처리

기계가 텍스트를 이해할 수 있도록 텍스트를 정제

신호와 소음을 구분

아웃라이어데이터로 인한 오버피팅을 방지

- HTML 태그, 특수문자, 이모티콘
- 정규표현식
- 불용어(Stopword)
- 어간추출(Stemming)
- 음소표기법(Lemmatizing)

정규화 normalization (입니다ㅋㅋ -> 입니다ㅋㅋ, 사랑해 -> 사랑해)

- 한국어를 처리하는 예시입니다ㅋㅋㅋㅋ -> 한국어를 처리하는 예시입니다ㅋㅋ

토큰화 tokenization

- 한국어를 처리하는 예시입니다ㅋㅋ -> 한국어Noun, 를Josa, 처리Noun, 하는Verb, 예시Noun, 입Adjective, 니다Eomi ㅋㅋ KoreanParticle

어근화 stemming (입니다 -> 이다)

- 한국어를 처리하는 예시입니다ㅋㅋ -> 한국어Noun, 를Josa, 처리Noun, 하다Verb, 예시Noun, 이다Adjective, ㅋㅋKoreanParticle

어구 추출 phrase extraction

- 한국어를 처리하는 예시입니다ㅋㅋ -> 한국어, 처리, 예시, 처리하는 예시

[출처 : twitter-korean-text] (<https://github.com/twitter/twitter-korean-text>)

불용어 Stopword

일반적으로 코퍼스에서 자주 나타나는 단어
학습이나 예측 프로세스에 실제로 기여하지 않음

예) 조사, 접미사 - 나, 너, 은, 는, 이, 가, 하다, 합니다 등

어간추출 Stemming

단어를 축약형으로 바꿔준다.

새로운 (like new), 새로울 (will be new)

→ 새롭다 (new)

먹었다 (ate), 먹을 (will eat), 먹을지도 모르는(may be eating)

→ 먹다 (eat)

[출처 : twitter-korean-text] (<https://github.com/twitter/twitter-korean-text>)

음소표기법 Lemmatization

품사정보가 보존된 형태의 기본형으로 변환

- 1) 배가 맛있다.
- 2) 배를 타는 것이 재미있다.
- 3) 평소보다 두 배로 많이 먹어서 배가 아프다.

영어에서 **meet**는 **meeting**으로 쓰였을 때 회의를 뜻하지만 **meet** 일 때는 만나다는 뜻을 갖는데 그 단어가 명사로 쓰였는지 동사로 쓰였는지에 따라 적합한 의미를 갖도록 추출하는 것

0과 1밖에 모르는 기계에게 인간의 언어 알려주기

컴퓨터는 숫자만 인식할 수 있기 때문에
바이너리 코드로 처리해 주기

텍스트 데이터 벡터화

One Hot Vector

텍스트 데이터, 범주형 데이터 => 수치형 데이터

- 머신러닝이나 딥러닝 알고리즘은 수치로된 데이터만 이해
- 벡터에서 해당되는 하나의 데이터만 1로 변경해 주고 나머지는 0으로 채워주는 것

과일	과일	과일_사과	과일_배	과일_감
사과	사과	1	0	0
배	배	0	1	0
감	감	0	0	1
사과	사과	1	0	0

BOW(Bag Of Words)

BOW (bag of words)

- 가장 간단하지만 효과적이라 널리쓰이는 방법
- 장, 문단, 문장, 서식과 같은 입력 텍스트의 구조를 제외하고 각 단어가 이 말뭉치에 얼마나 많이 나타나는지만 헤아림
- 구조와 상관없이 단어의 출현횟수만 세기 때문에 텍스트를 담는 가방 (bag)으로 생각할 수 있음
- BOW는 단어의 순서가 완전히 무시 된다는 단점
- 예를 들어 의미가 완전히 반대인 두 문장을 보자
 - it's bad, not good at all.
 - it's good, not bad at all.
- 위 두 문장은 의미가 전혀 반대지만 완전히 동일하게 반환
- 이를 보완하기 위해 n-gram을 사용하는 데 BOW는 하나의 토큰을 사용하지만 n-gram은 n개의 토큰을 사용

BOW(bag of words)

- (1) 동물원에 버스를 타고 갔다.
- (2) 동물원에서 사자를 봤다.
- (3) 동물원에서 기린을 봤다.

토큰화

[
"동물원에", "동물원에서", "버스를", "사자를", "기린을", "타고", "갔다", "봤다"
]

벡터화

- (1) [1, 0, 1, 0, 0, 1, 1, 0]
- (2) [0, 1, 0, 1, 0, 0, 0, 1]
- (3) [0, 1, 0, 0, 1, 0, 0, 1]

n-gram 동물원에 버스를 타고 갔다.

uni-gram

```
[  
"동물원에",  
"버스를"  
"타고",  
"갔다",  
]
```

bi-gram

```
[  
"동물원에 버스를",  
"버스를 타고",  
"타고 갔다"  
]
```

tri-gram

```
[  
"동물원에 버스를 타고",  
"버스를 타고 갔다"  
]
```

bi-gram(1,2)

```
[  
"동물원에",  
"버스를",  
"타고갔다",  
"동물원에 버스를",  
"버스를 타고",  
"타고 갔다"  
]
```

tri-gram(2,3)

```
[  
"동물원에 버스를",  
"버스를 타고",  
"타고 갔다",  
"동물원에 버스를 타고",  
"버스를 타고 갔다"  
]
```

TF-IDF

Term frequency Inverse document frequency

TF(단어 빈도, term frequency)는 특정한 단어가 문서 내에 얼마나 자주 등장하는지를 나타내는 값

이 값이 높을수록 문서에서 중요하다고 생각할 수 있지만
단어 자체가 문서군 내에서 자주 사용되는 경우,
이것은 그 단어가 흔하게 등장한다는 것을 의미

이것을 **DF(문서 빈도, document frequency)**라고 하며,

이 값의 역수를 **IDF(역문서 빈도, inverse document frequency)**라고 함

TF-IDF는 TF와 IDF를 곱한 값

Word2Vec

Word Embedding to Vector

Word2Vec

one hot encoding(예 [0000001000]) 혹은 Bag of Words에서
vector size가 매우 크고 sparse 하므로 neural net 성능이 잘 나오지 않음

- 주위 단어가 비슷하면 해당 단어의 의미는 유사하다 라는 아이디어
- 단어를 트레이닝 시킬 때 주변 단어를 label로 매치하여 최적화
- 단어를 의미를 내포한 dense vector로 매칭 시키는 것

Word2Vec은 분산 된 텍스트 표현을 사용하여 개념 간 유사성을 봄
예를 들어, 베이징과 중국이 서울과 한국이 (수도와 나라) 같은 방식으로 관련되어 있음을
이해

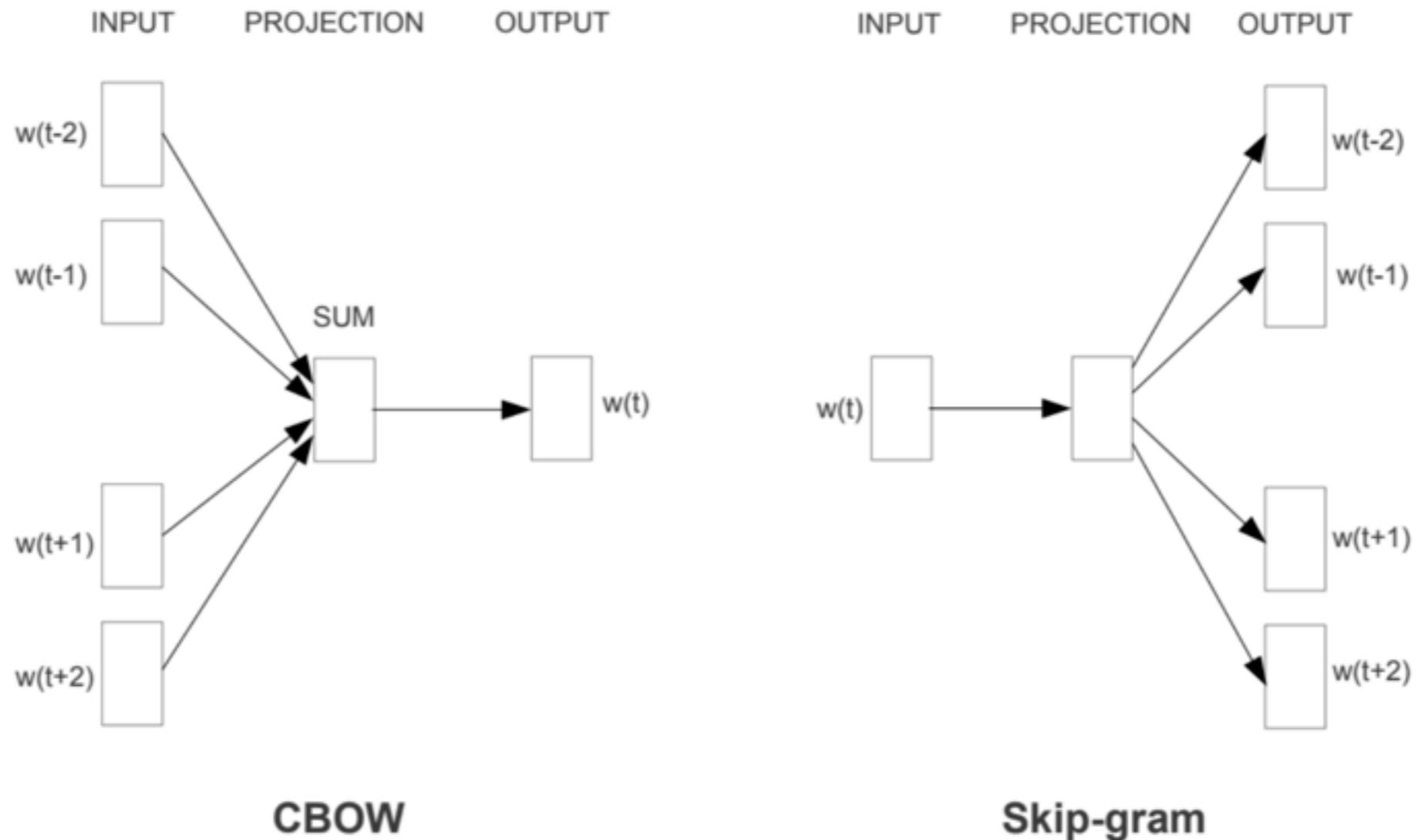


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

CBOW (continuous bag-of-words)

전체 텍스트로 하나의 단어를 예측하기 때문에 작은 데이터셋일 수록 유리하다.
아래 예제에서 _에 들어갈 단어를 예측한다.

- 1) _가 맛있다.
- 2) _를 타는 것이 재미있다.
- 3) 평소보다 두 _로 많이 먹어서 _가 아프다.

Skip-Gram

타겟 단어들로부터 원본 단어를 역으로 예측하는 것이다. CBOW와는 반대로 컨텍스트-타겟 쌍을 새로운 발견으로 처리하고 큰 규모의 데이터셋을 가질 때 유리

배라는 단어 주변에 올 수 있는 단어를 예측한다.

- 1) *배*가 맛있다.
- 2) *배*를 타는 것이 재미있다.
- 3) 평소보다 두 *배*로 많이 먹어서 *배*가 아프다.

청와대 국민청원 데이터

- Word2Vec을 통해 벡터화된 데이터로 단어 유사도 보기

```
# 가장 유사한 단어를 추출
```

```
model.wv.most_similar('대통령')
```

```
[('청와대', 0.9971275925636292),  
 ('강화', 0.996383786201477),  
 ('박근혜', 0.9961060285568237),  
 ('대통령은', 0.9960640668869019),  
 ('정부', 0.995932400226593),  
 ('대통령의', 0.9959185123443604),  
 ('드렸습니다', 0.9948124289512634),  
 ('이명박', 0.9947943687438965),  
 ('교육비를', 0.9945780634880066),  
 ('건의', 0.9941753149032593)]
```

```
# 유사도가 없는 단어 추출
```

```
model.wv.doesnt_match('이명박 박근혜 대통령 김정은'.split())
```

'김정은'

```
# 유사도가 없는 단어 추출
```

```
model.wv.doesnt_match('이명박 박근혜 대통령 김정은 문재인'.split())
```

'문재인'

단어 유사도

```
model.wv.similarity('블록체인', '가상화폐')
```

0.994049599379707

```
model.wv.similarity('블록체인', '현금')
```

0.9836642661916147

```
model.wv.similarity('블록체인', '주식')
```

0.985456522103714

```
model.wv.similarity('블록체인', '종이')
```

0.9487120361290824

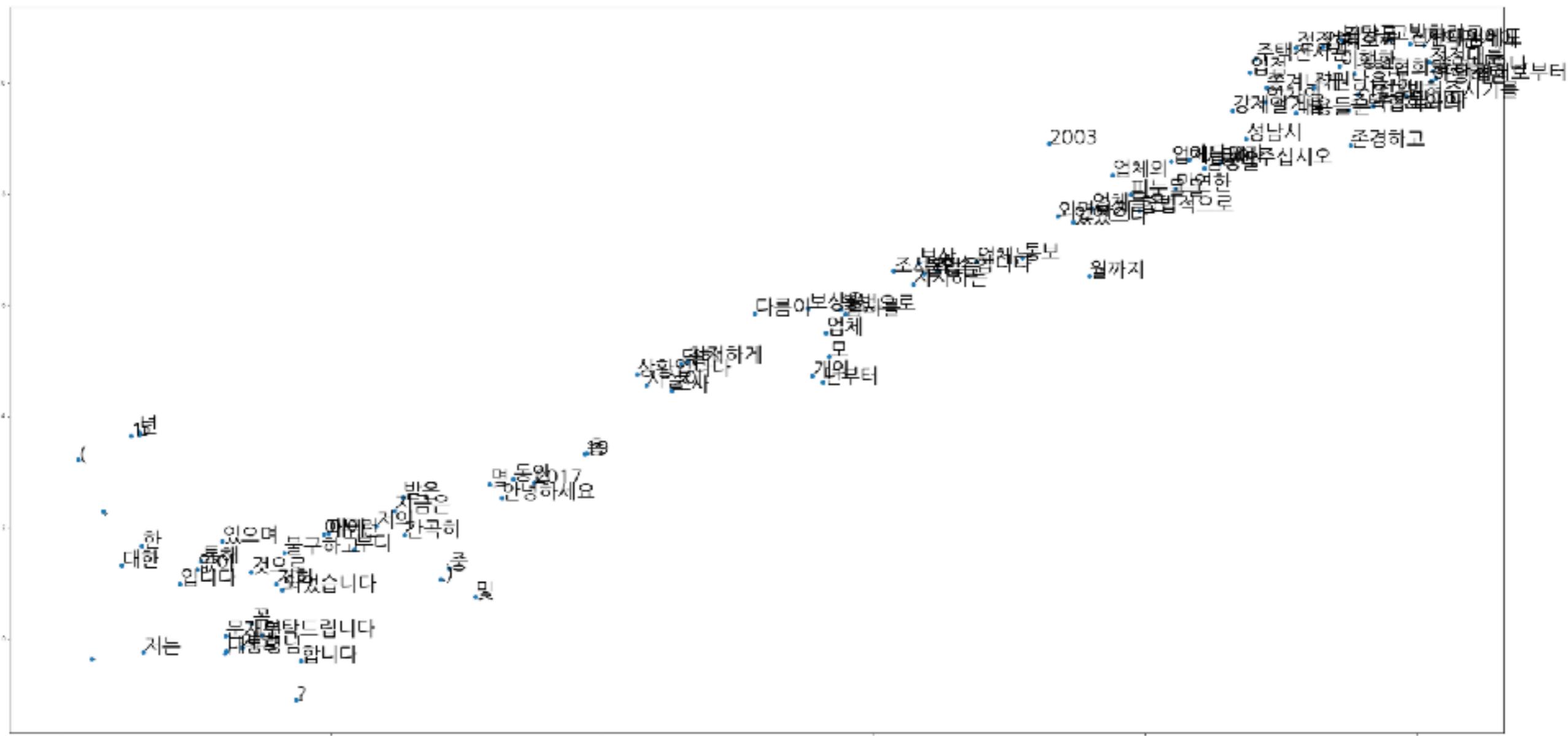
텍스트 데이터 시각화

청와대 국민청원 데이터로 그린 워드클라우드



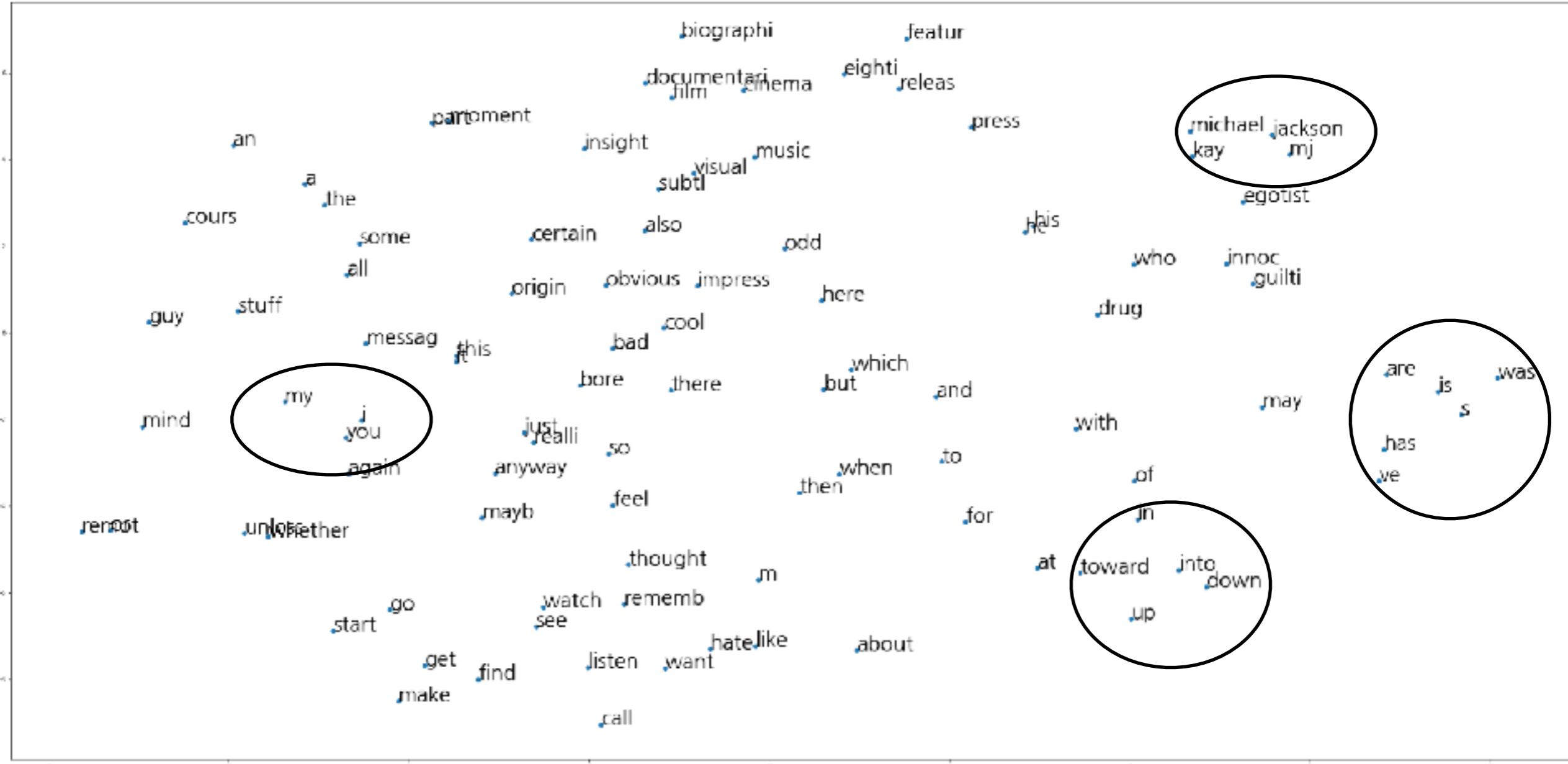
청와대 국민청원 데이터

- Word2Vec으로 벡터화하고 일부 데이터를 차원축소 기법으로 줄여서 표현
 - 비슷한 단어끼리 비슷한 위치에 분포(예. 간곡히, 부탁드립니다)
 - 조사와 불용어가 섞여있어서 데이터 정제가 필요

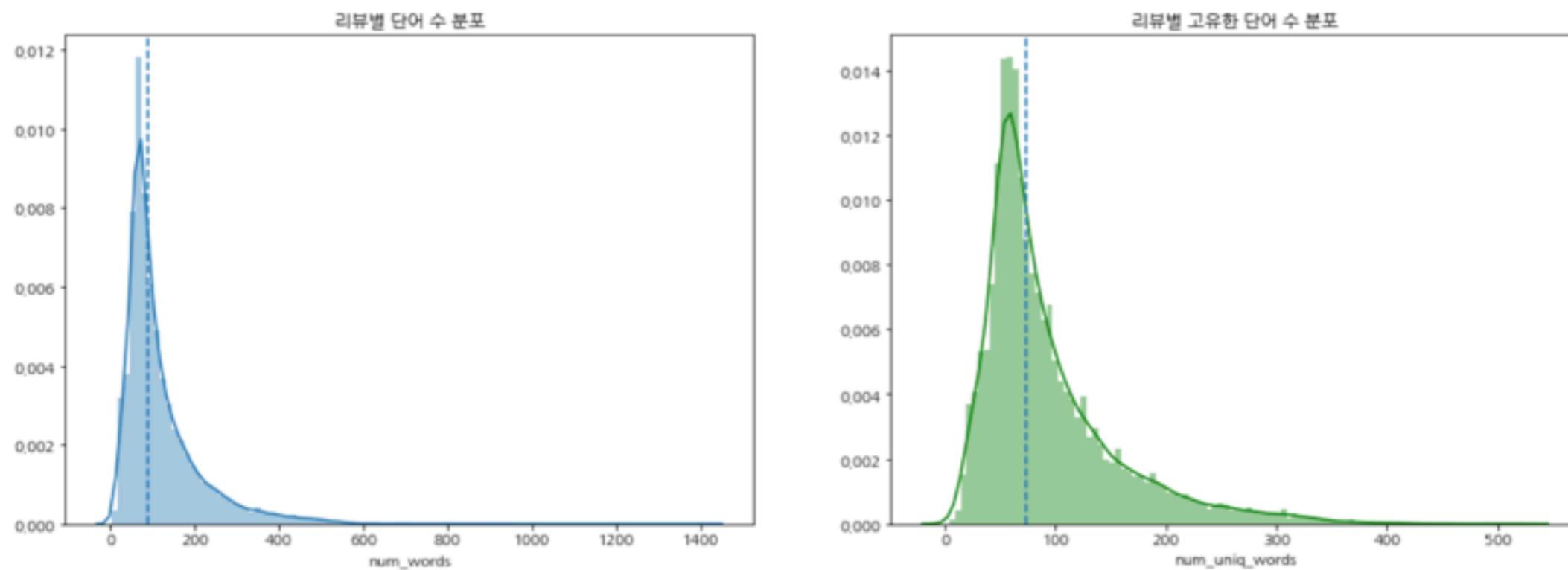


IMDB 영화리뷰 데이터

- Word2Vec으로 벡터화하고 일부 데이터를 차원축소 기법으로 줄여서 표현
- [my, i, u], [toward, into, up, down]이 비슷한 위치

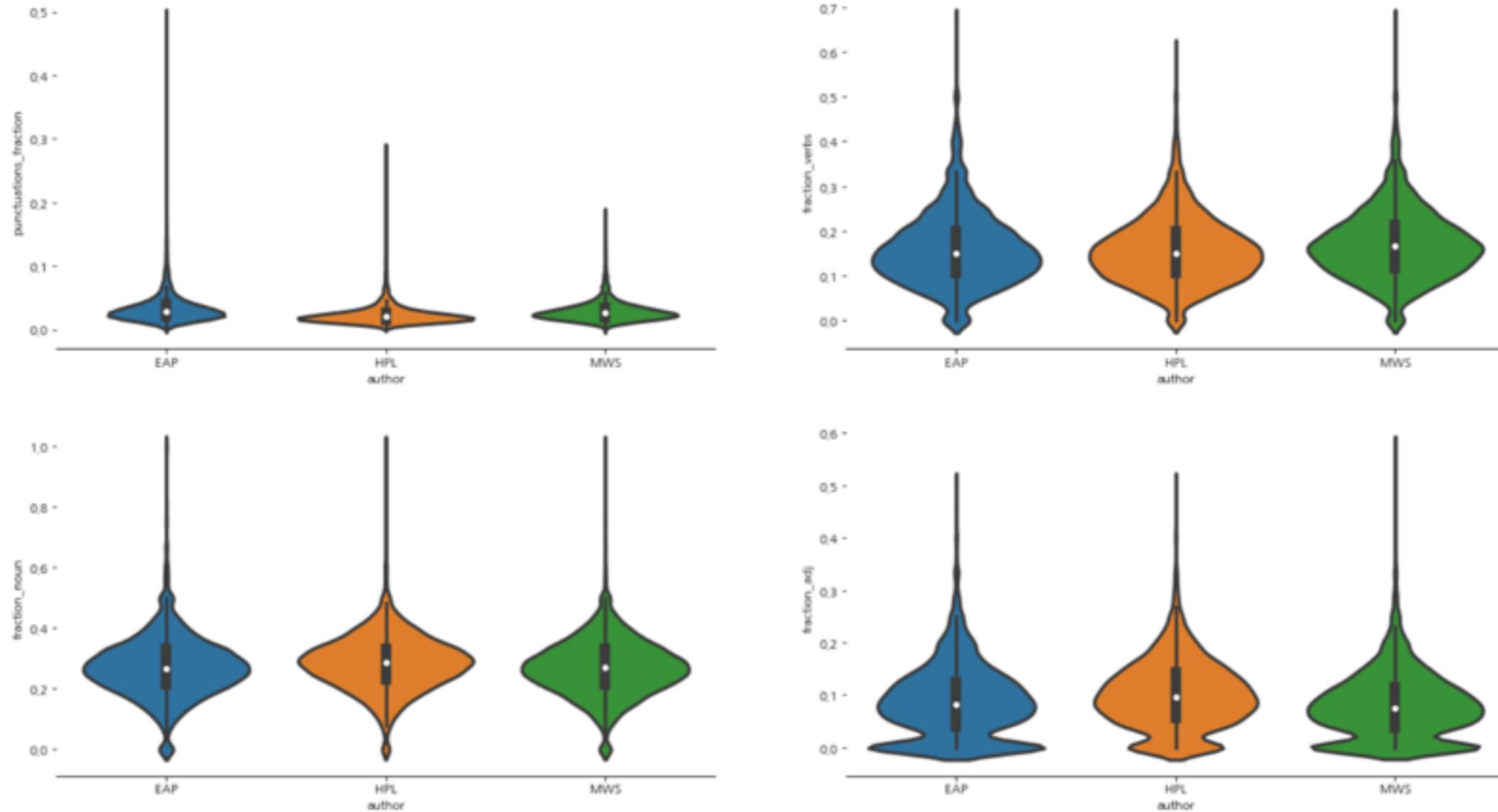


단어 수 혹은 문장길이, 특수문자, 불용어 갯수 등을 시각화



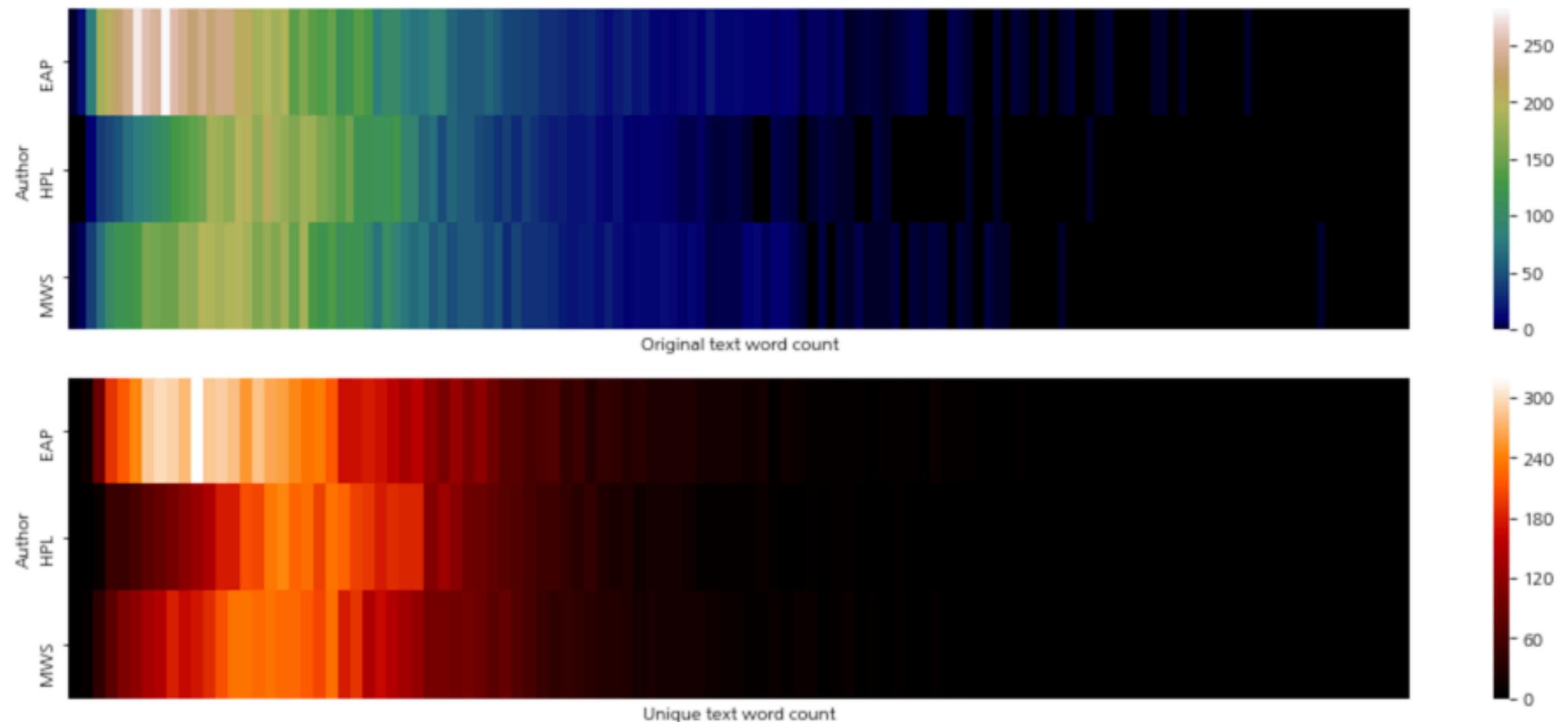
IMDB 영화리뷰의 단어 수 분포

작가별 품사 사용에 대한 시각화



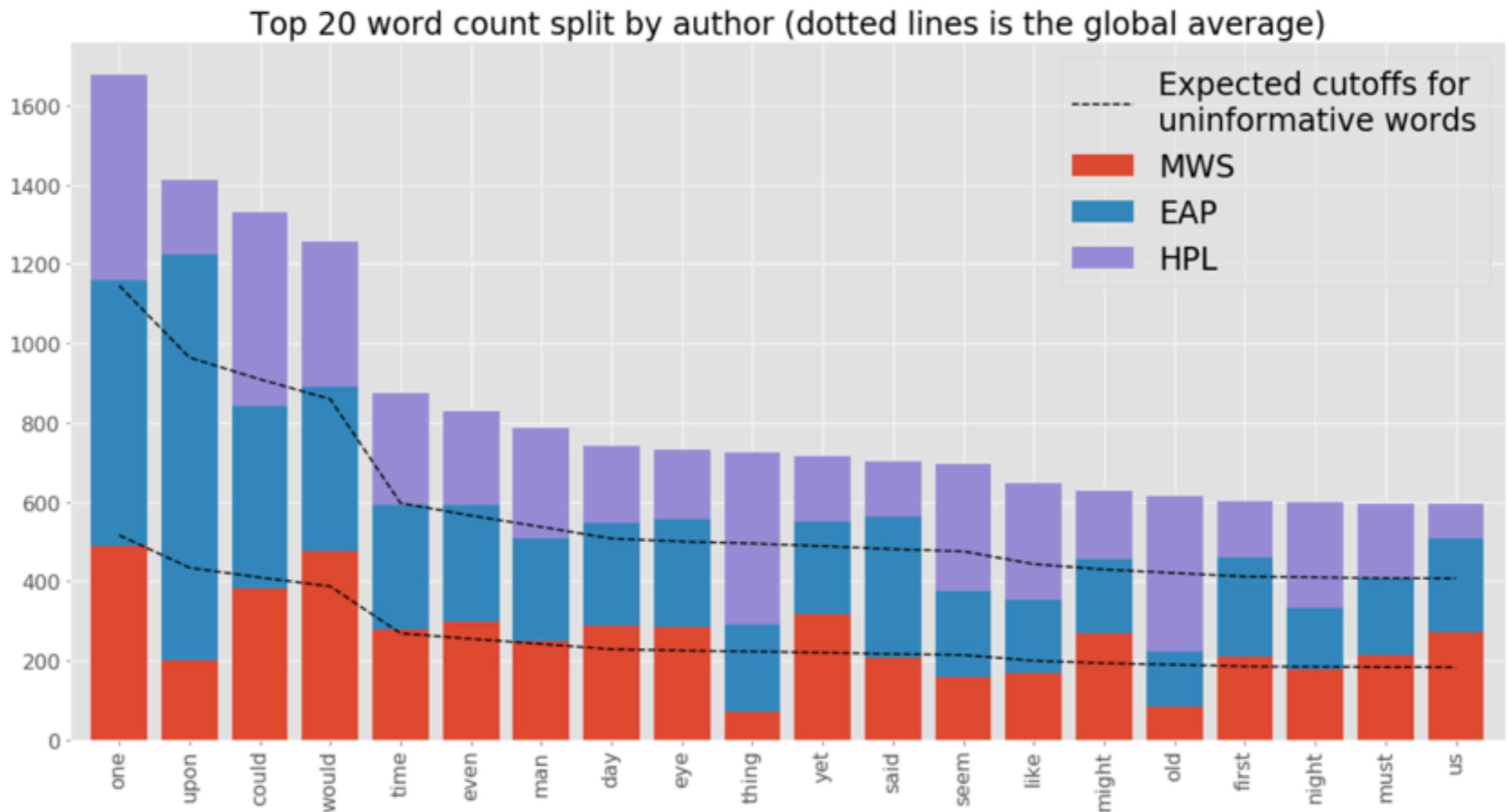
사용 데이터셋 : <https://www.kaggle.com/c/spooky-author-identification>

작가별 단어개수 시각화



사용 데이터셋 : <https://www.kaggle.com/c/spooky-author-identification>

작가별로 소설에 자주 등장하는 단어



출처 : <https://www.kaggle.com/marcospinaci/talking-plots-2-adding-grammar>

사용 데이터셋 : <https://www.kaggle.com/c/spooky-author-identification>

그럼 자연어 처리로 어떤 일을 할 수 있을까?

맞춤법 수정

자동요약

스팸메일 검출

기계번역

자동답변

분류

챗봇

고객센터

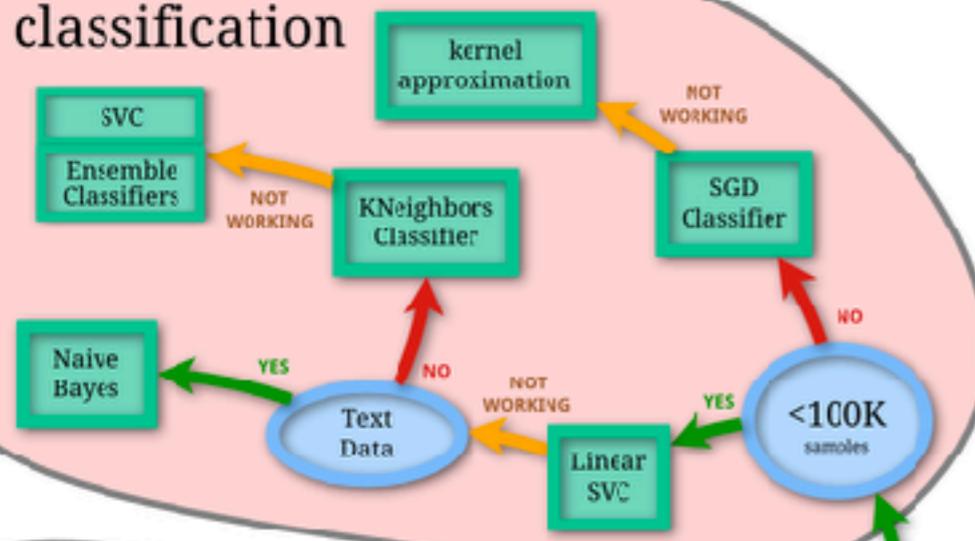
추천

감정분석

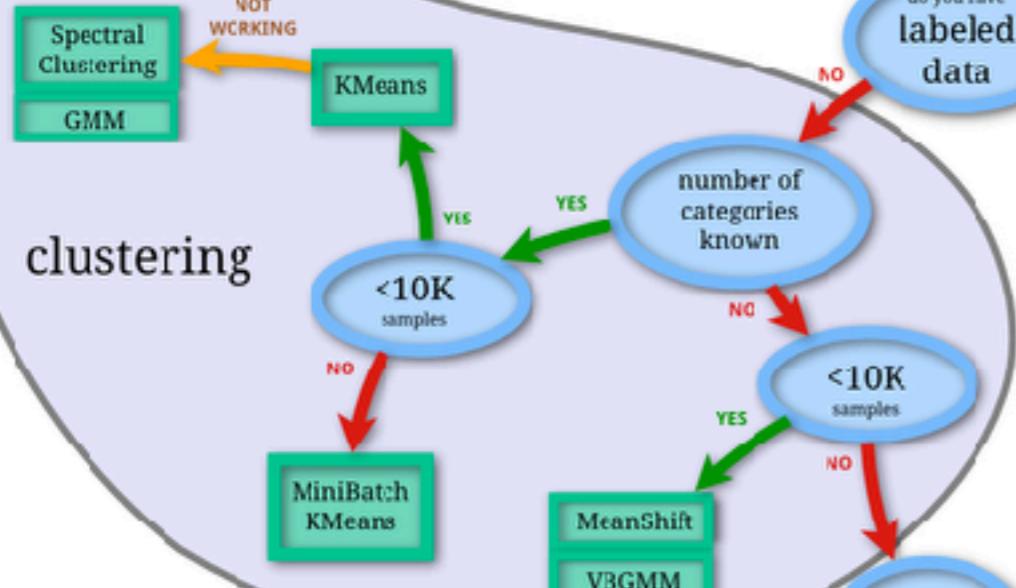
자연어 처리로 할 수 있는 일

스팸메일 분류

classification



clustering

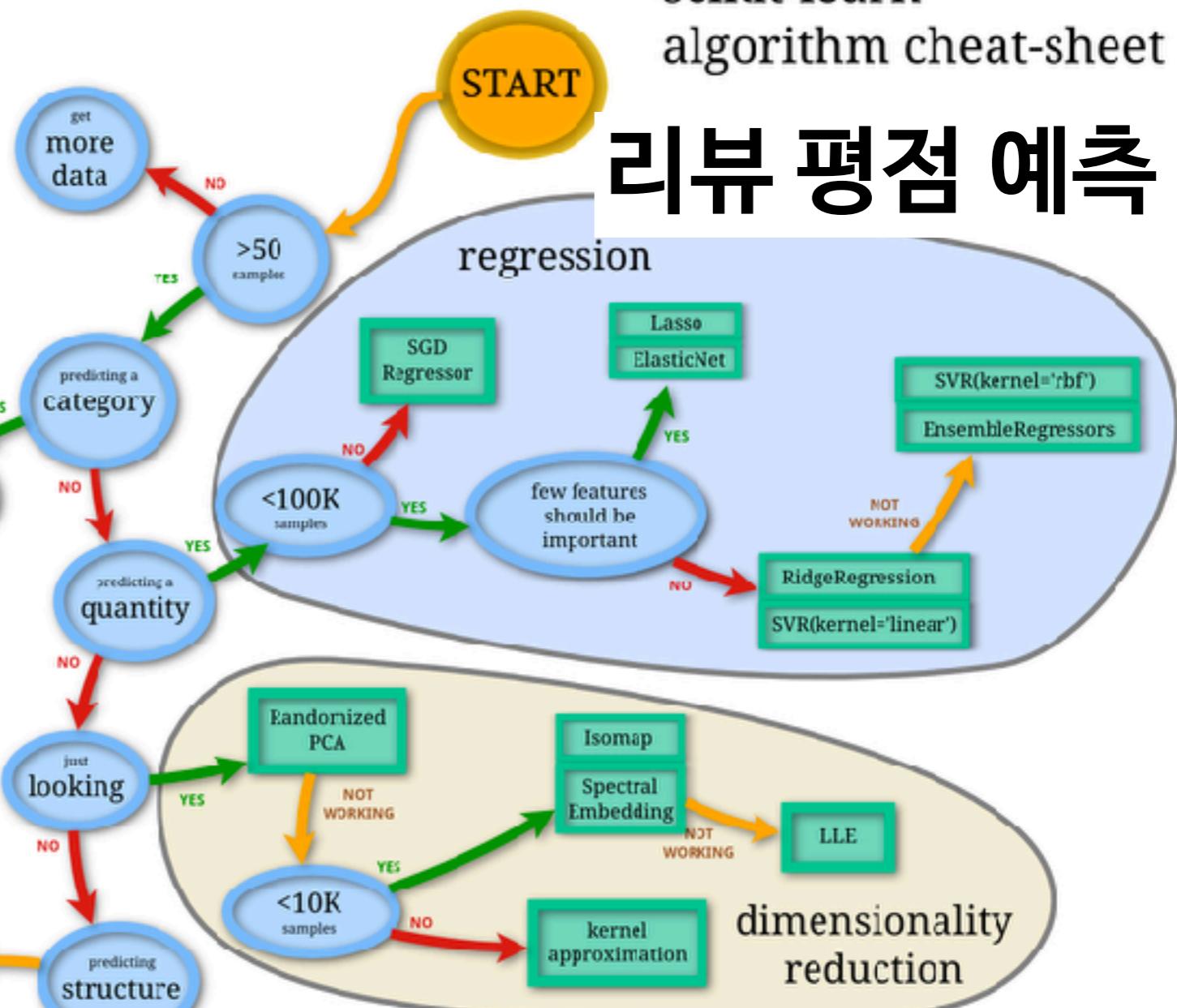


비슷한 메일끼리 모으기

scikit-learn
algorithm cheat-sheet

리뷰 평점 예측

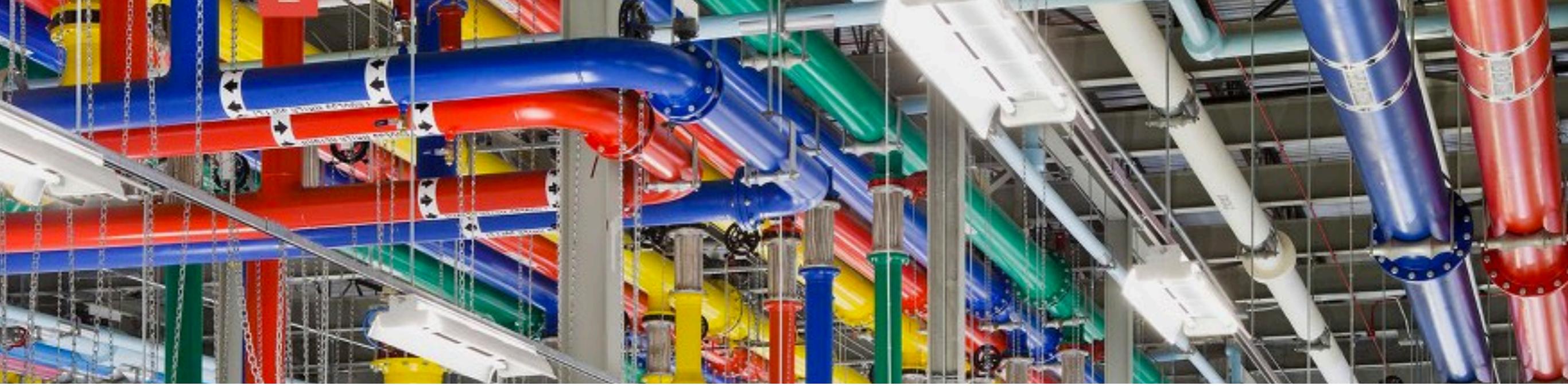
regression



차원 축소 기법으로 시각화

자연어 처리에서 활용하기





감사합니다.

