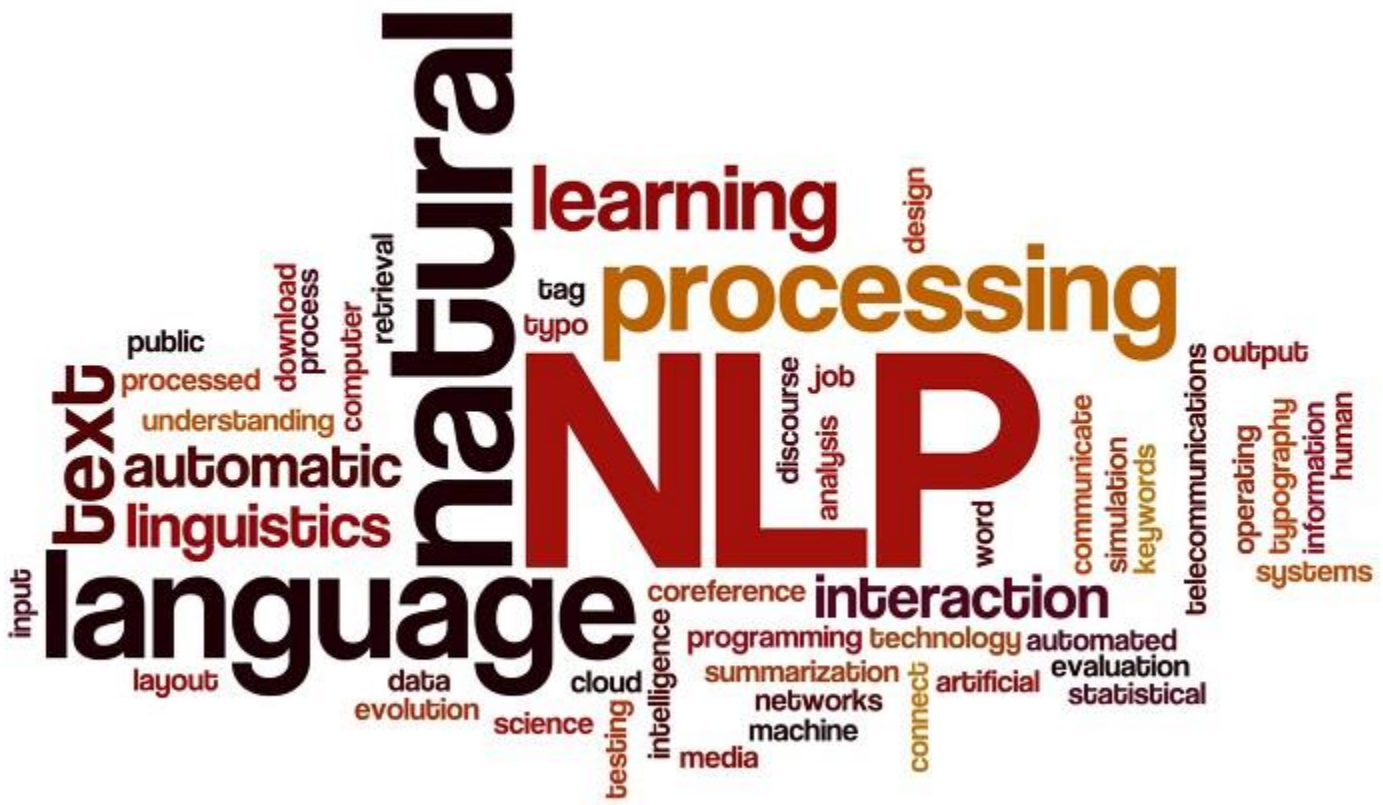
19기 정규세션

ToBig's 18기 강의자
현승현

Week 7:
NLP Basic

Contents

Natural Language Processing



Contents

Corpus : 대량의 텍스트 집합

코퍼스 수집

전처리
(preprecessing)

문장 분절

토큰화
(Tokenize)

어간 추출
(Stemming)

표제어 추출
(Lemmatize)

코퍼스 수집

전처리
(preprecessing)

정규표현식 사용

- HTML 태그 제거
- 특수문자 제거

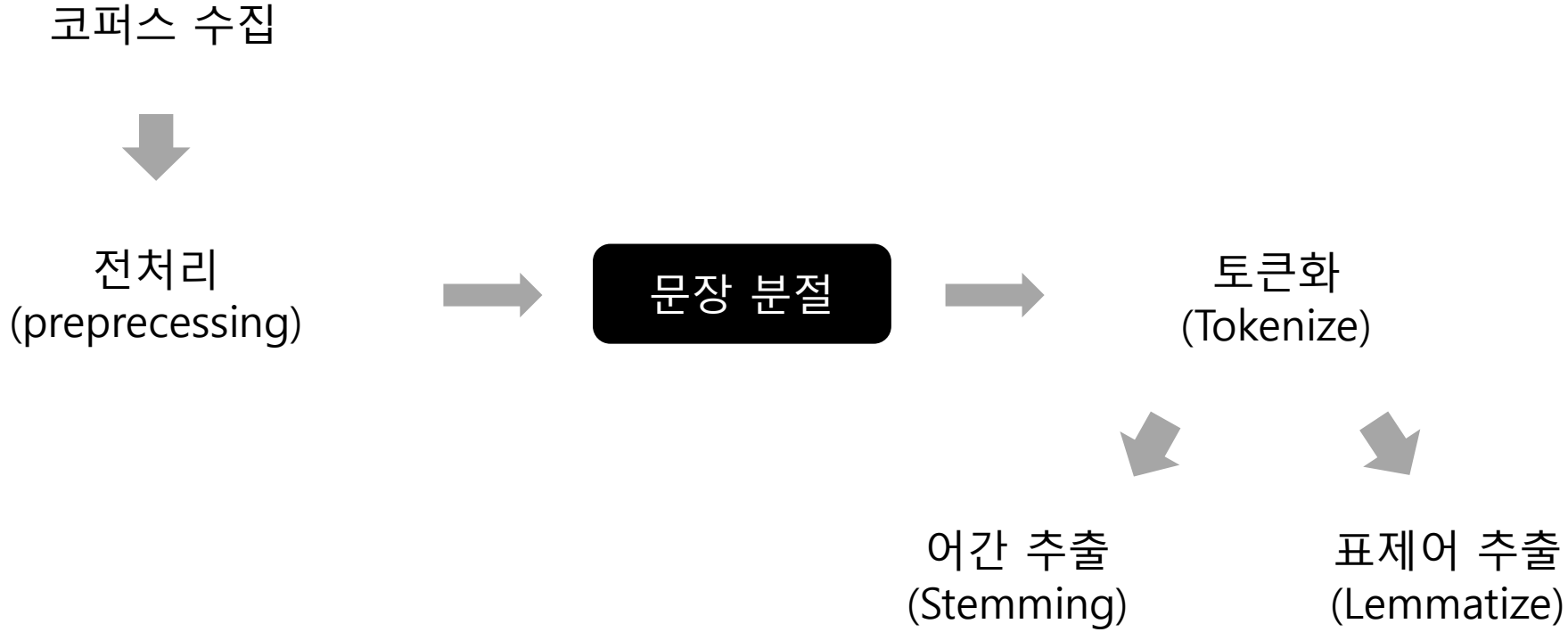
ex. 초등학교 입학을 축하합니다.~~^^;

문장 분절

토큰화
(Tokenize)

어간 추출
(Stemming)

표제어 추출
(Lemmatize)


코퍼스 수집
전처리
(preprecessing)
문장 분절
토큰화
(Tokenize)
어간 추출
(Stemming)
표제어 추출
(Lemmatize)

문장 분절 (Sentence Tokenization)

- 대부분의 NLP Task는 입력 단위가 **'문장'**이기에 문장 단위로 분절해주는 절차가 필요하다.
- 하지만, 단순히 마침표를 기준으로 문장을 분절할 경우 잘못 분절될 수 있다. (ex. 약어, 소수점, 네트워크 주소 등)
- 따라서, nltk, kss, genia 등에서 제공하는 **문장 분절 모듈**을 사용해야한다.

[문장 분리 예제]

```
[ ] from nltk.tokenize import sent_tokenize
```

```
[ ] text="""His barber kept his word.
      But keeping such a huge secret to himself was driving him crazy.
      Finally, the barber went up a mountain and almost to the edge of a cliff.
      He dug a hole in the midst of some reeds.
      He looked about, to mae sure no one was near.
      """
```

```
[ ] sent_tokenize(text)

['His barber kept his word.',
 'But keeping such a huge secret to himself was driving him crazy.',
 'Finally, the barber went up a mountain and almost to the edge of a cliff.',
 'He dug a hole in the midst of some reeds.',
 'He looked about, to mae sure no one was near.']
```

```
[ ] text2="I am actively looking for Ph.D. students. You are not a Ph.D student. Sorry."
```

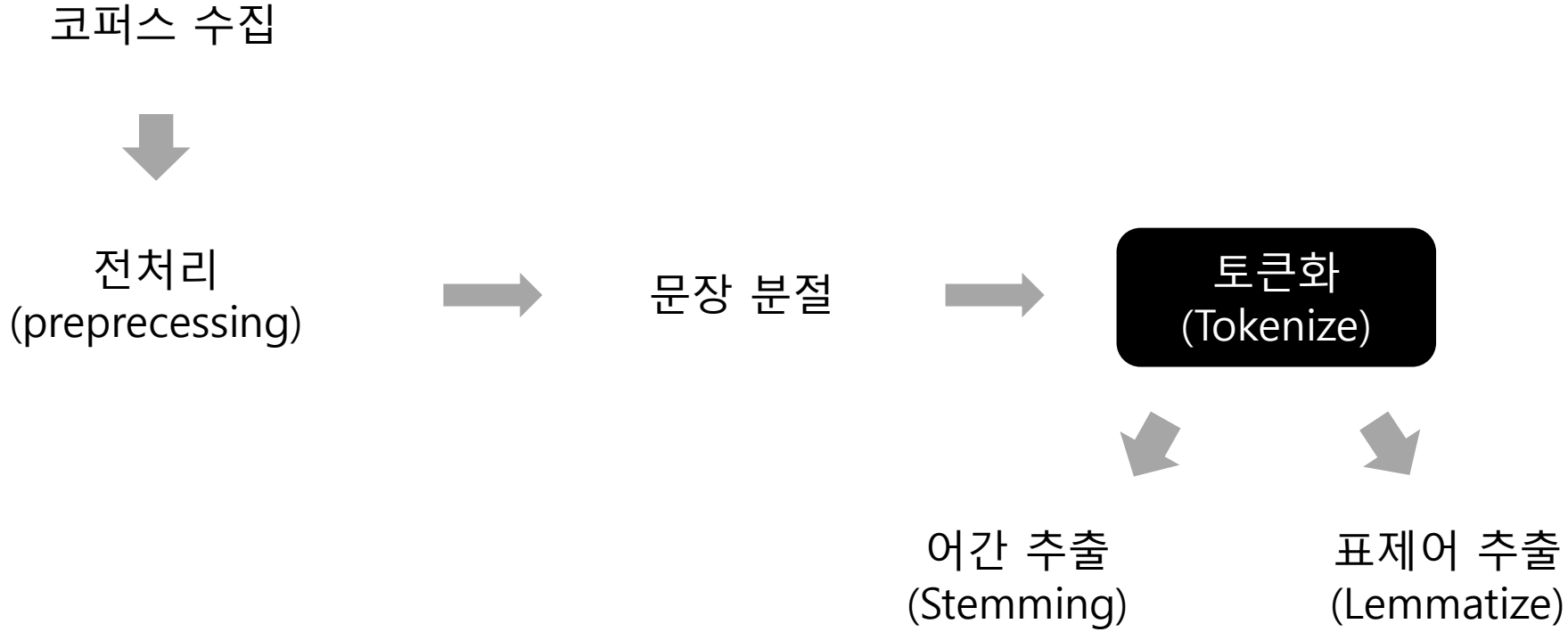
```
[ ] sent_tokenize(text2)

['I am actively looking for Ph.D. students.',
 'You are not a Ph.D student.',
 'Sorry.']
```

```
[ ] text3 = "Network prefix: IP 255.255.255.0. Network Address number is 192.168.56.31. Thanks!"
```

```
[ ] sent_tokenize(text3)

['Network prefix: IP 255.255.255.0.',
 'Network Address number is 192.168.56.31.',
 'Thanks!']
```


코퍼스 수집
전처리
(preprecessing)
문장 분절
토큰화
(Tokenize)
어간 추출
(Stemming)
표제어 추출
(Lemmatize)

토큰화 (Tokenization)

- 토큰화 역시 모듈을 사용해 쉽게 할 수 있다.
- 하지만 각 모듈별로 마침표, 아포스트로피 등을 처리하는 방법이 다르기 때문에 적절히 선택해야 한다.

[영어 토큰화 예제(nltk)]

```
from nltk.tokenize import TreebankWordTokenizer, WordPunctTokenizer, word_tokenize
from nltk.tokenize.moses import MosesTokenizer
```

word_tokenize	I	am	actively	looking	for	Ph.D.	students	.	None	None	None
WordPunctTokenizer	I	am	actively	looking	for	Ph	.	D	.	students	.
TreebankWordTokenizer	I	am	actively	looking	for	Ph.D.	students	.	None	None	None
MosesTokenizer	I	am	actively	looking	for	Ph.D	.	students	.	None	None

토큰화 (Tokenization)

[영어 토큰화 예제(nltk)]

word_tokenize	Network	Address	number	is	192.168.56.31	.	None	None	None	None	None	None
WordPunctTokenizer	Network	Address	number	is	192	.	168	.	56	.	31	.
TreebankWordTokenizer	Network	Address	number	is	192.168.56.31	.	None	None	None	None	None	None
MosesTokenizer	Network	Address	number	is	192.168.56.31	.	None	None	None	None	None	None

word_tokenize	I	'll	be	back	as	soon	as	possible	. None
WordPunctTokenizer	I	'	ll	be	back	as	soon	as	possible .
TreebankWordTokenizer	I	'll	be	back	as	soon	as	possible	. None
MosesTokenizer	I	'll	be	back	as	soon	as	possible	. None

토큰화 (Tokenization)

- 한국어의 경우 영어보다 복잡하다.
- 이는 한국어가 교착어(어근+접사)이기 때문이다.

고양이가 한 마리 있다. ➡ "고양이", "가", "한", "마리", "있다"

[한국 토큰화 예제(konlpy)]

```
from konlpy.tag import Okt, Kkma
```

Okt	열심히	코딩	하지	않은	당신	,	조금	더	열심히	해보는	것	은	어떨까	요	?	None	None	None
Kkma	열심히	코딩	하	지	않	은	당신	,	조금	더	열심히	해보	는	것	은	어떻	ㄹ까요	?

Okt	IP	192.168	.	56.31	서버	에	들어가서	로그	파일	저장	해서	tobigs1516@gmail.com	로	결과	좀	보내줘	.	None	None	None	None	None	None	None
Kkma	IP	192.168	.	56.31	서버	에	들어가	서	로그	파일	저장해	서	tobigs	1516	@	gmail	.	com	로	결과	좀	보내주	어	.

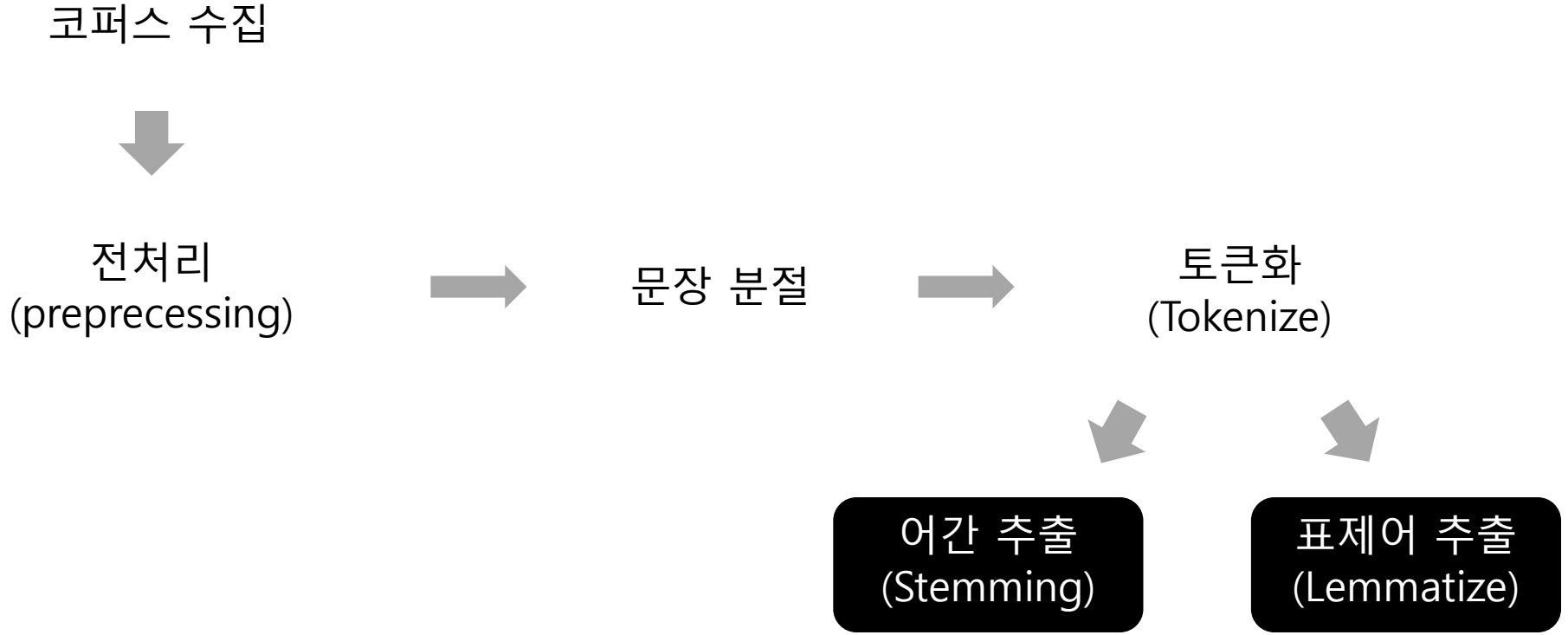
품사 태깅(Part-of-speech Tagging)

- 단어 표기는 같지만, 품사에 따라 단어의 의미가 달라지기도 한다. Ex) fly → 명사('파리') / 동사('날다')
- 단어의 의미를 제대로 파악하기 위해서 해당 단어가 어떤 품사로 쓰였는지 보는 것이 중요할 수 있다.
- 품사태깅 역시 모듈을 사용해 자동으로 품사를 태깅할 수 있다.

[한국 품사 태깅 예제(konlpy)]

```
from konlpy.tag import Okt, Kkma, Hannanum
```

okt	(한국, Noun)	(은, Josa)	(지난, Noun)	(2년, Number)	(동안, Noun)	(아시아, Noun)	(지역, Noun)	(을, Josa)	(휩쓴, Adjective)	(경제, Noun)	(적, Suffix)	(위기, Noun)	(를, Josa)	(국민, Noun)	(과, Josa)	(정부, Noun)	(의, Josa)	(헌신, Noun)	(., Punctuation)	(그리고, Conjunction)
kkma	(한국, NNG)	(은, JX)	(지나, VV)	(ㄴ, ETD)	(2, NR)	(년, NNM)	(동안, NNG)	(아시아, NNG)	(지역, NNG)	(을, JKO)	(휩쓸, VV)	(ㄴ, ETD)	(경제적, NNG)	(위기, NNG)	(를, JKO)	(국민, NNG)	(과, JC)	(정부, NNG)	(의, JKG)	(헌신, NNG)
hannanum	(한국, N)	(은, J)	(지나, P)	(ㄴ, E)	(2년, N)	(동안, N)	(아시아, N)	(지역, N)	(을, J)	(휩쓸, P)	(ㄴ, E)	(경제적, N)	(위, N)	(이, J)	(기, E)	(를, J)	(국민, N)	(과, J)	(정부, N)	(의, J)


코퍼스 수집
전처리
(preprecessing)
문장 분절
토큰화
(Tokenize)
어간 추출
(Stemming)
표제어 추출
(Lemmatize)

어간 추출(Stemming) & 표제어 추출 (Lemmatization)

- 눈으로 봤을 때는 서로 다른 단어이지만, 하나의 단어로 일반화시킬 수 있다면 일반화시켜서 문서 내의 단어 수를 줄이고자 할 때 사용한다.
- 보통 단어의 빈도수를 기반으로 하는 BoW(Bag of Words) 표현을 사용하는 자연어 처리 문제에서 사용한다.
- 자연어 처리에서 전처리, 더 정확히는 정규화의 지향점은 언제나 갖고 있는 코퍼스로부터 복잡성을 줄이는 일이다.

표제어 추출(Lemmatization)

- 표제어(Lemma)는 '표제어' 또는 '기본 사전형 단어' 정도의 의미를 갖는다.
- 즉, 표제어 추출은 단어로부터 표제어를 찾아가는 과정이다.
- 표제어 추출을 하는 가장 섬세한 방법은 단어의 형태학적 파싱을 먼저 진행하는 것이다.
- 형태학적 파싱은 형태소에서 어간과 접사를 분리하는 작업을 말한다.

의미를 가진 가장 작은 단위

단어의 의미를 담고 있는 핵심 부분

단어에 추가적인 의미를 주는 부분

Ex.

cats → cat

am → be

having → have

어간 추출(Stemming)

- 어간 추출을 형태학적 분석을 단순화한 버전이라고 볼 수도 있고, 정해진 규칙만 보고 단어의 어미를 자르는 작업이라고 볼 수도 있다.
- 이 작업은 섬세한 작업이 아니기 때문에, 결과 단어가 사전에 존재하지 않는 단어일 수도 있다.

Ex.
This → Thi

was → wa

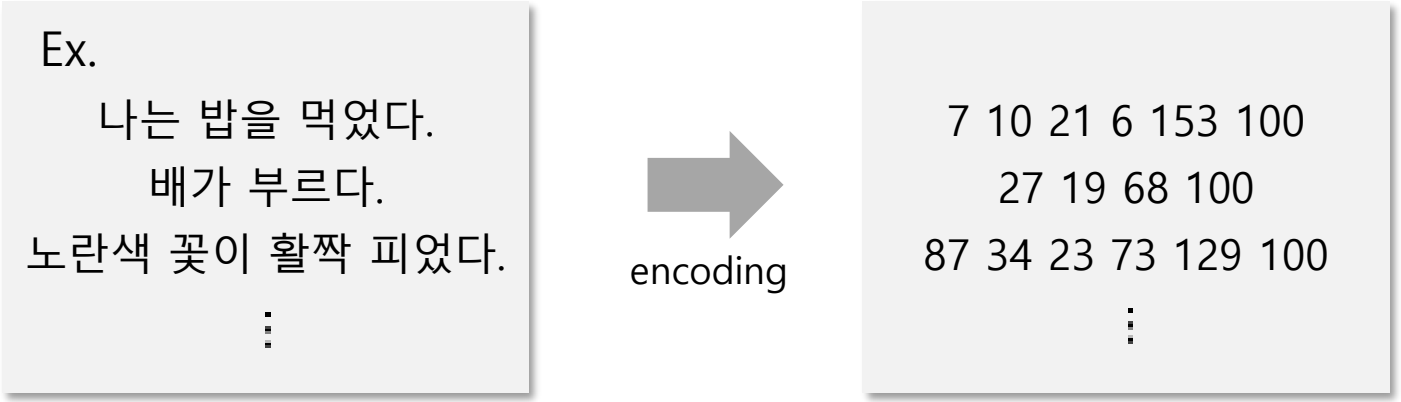
```
from nltk.stem import PorterStemmer, LancasterStemmer, WordNetLemmatizer
```

Token	There	was	no	hope	for	him	this	time	:	it	was	the	third	stroke	.	Night	after	night	I	had	passed	the	house	and	studied	the	lighted
PorterStemmer()	there	wa	no	hope	for	him	thi	time	:	it	wa	the	third	stroke	.	night	after	night	I	had	pass	the	hous	and	studi	the	light
LancasterStemmer()	ther	was	no	hop	for	him	thi	tim	:	it	was	the	third	stroke	.	night	aft	night	i	had	pass	the	hous	and	study	the	light
WordNetLemmatizer()	There	wa	no	hope	for	him	this	time	:	it	wa	the	third	stroke	.	Night	after	night	I	had	passed	the	house	and	studied	the	lighted

Token	don't	doing	have	has	going	goes	gone	went	puppies	living	lives	fly	mainly	dies	watched	starting	policy	organization
LancasterStemmer()	don't	do	have	ha	go	goe	gone	went	puppi	live	live	fli	mainli	die	watch	start	polici	organ
WordNetLemmatizer()	don't	doing	have	ha	going	go	gone	went	puppy	living	life	fly	mainly	dy	watched	starting	policy	organization

Integer Encoding(정수 인코딩)

- 컴퓨터가 자연어를 처리하기 위해선, 자연어를 컴퓨터가 이해할 수 있는 숫자로 바꿔줘야 한다.
- 자연어 처리에는 텍스트를 숫자로 바꾸는 여러가지 기법들이 있고, 이러한 기법들을 본격적으로 적용하기 위한 첫 단계로 각 단어(토큰)을 고유한 정수에 맵핑(mapping)시키는 전처리 작업이 필요하다.
- 고유한 정수, 즉 인덱스를 부여할 때 랜덤으로 부여하기도 하지만, 보통 단어 등장 빈도수를 기준으로 정렬한 뒤에 부여한다.



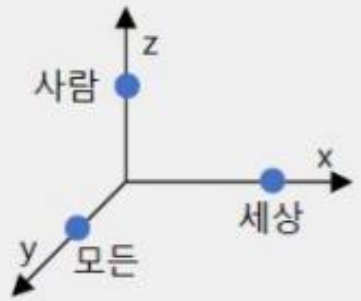
One-hot Encoding

- 자연어를 컴퓨터가 인지할 수 있는 수치로 바꾸는 가장 간단한 방법
- 각 토큰에 배정된 인덱스에 1, 나머지를 0으로 표현하는 방법
- 차원 == 단어 집합 크기 *단어 집합은 중복 허용하지 않고 서로 다른 단어들의 집합

세상 모든 사람

단어	Vector
세상	[1, 0, 0]
모든	[0, 1, 0]
사람	[0, 0, 1]

n개의 단어는 n차원의 벡터로 표현



One-hot Encoding

- **차원의 저주**
  - 단어의 개수가 늘어날수록, 벡터를 저장하기 위해 필요한 공간이 계속 늘어난다.
  - 같은 크기의 공간에 표현되는 정보의 양보다 적다(sparse representation)
- **단어의 의미 정보 표현 불가**
  - 벡터 연산 시 결과값이 0이 되어 similarity 계산 불가
  - 이상적으로는 단어들 사이의 유사성이 높으면 가까이 위치해야 한다.
  - 하지만, one-hot encoding에서는 모든 단어들 사이의 거리가 일정하다.

단어의 **의미**를 반영하면서도, **Dense한 낮은 차원**의 벡터를 만들어보자!

* 단어의 잠재 의미를 반영하여 다차원 공간에 벡터화 하는 기법
Word Embedding

1) 카운트 기반의 벡터화 방법 : LSA(잠재 의미 분석), HAL
2) 예측 기반의 벡터화 방법 : Word2Vec, FastText
3) 2가지 방법을 모두 사용하는 벡터화 방법 : GloVe

Contents

언어 모델(Language Model)

- 언어 모델(Language Model)이란 **단어 시퀀스(문장)에 확률을 할당**하는 모델이다.
- 즉, 이전 단어들이 주어졌을 때 다음 단어를 예측하도록 하는 것입니다.
- 언어 모델을 만드는 방법은 크게는 통계를 이용한 방법과 인공 신경망을 이용한 방법으로 구분할 수 있습니다.

- **단어 시퀀스에 확률을 할당하는 것이 "왜" 필요한가**
  - 기계 번역(Machine Translation)

  : P("나는 버스를 탔다") > P("나는 버스를 태운다")

  - 오타 교정(Spell Correction)

  : "퇴근하고 집으로 버스를 _____"

  P("타고 간다") > P("달려 간다")

  → 언어 모델은 두 문장을 비교하여 좌측의 문장의 확률이 더 높다고 판단함으로써 여러 후보들 중 더 자연스러운 선택지를 선택한다.

언어 모델(Language Model)

- **주어진 이전 단어들로부터 다음 단어를 "어떻게" 예측할 것인가**

  하나의 단어를 w, 단어 시퀀스를 W라고 한다면, n개의 단어가 등장하는 단어 시퀀스 W의 확률은 다음과 같다.

$$P(W) = P(w_1, w_2, w_3, w_4, w_5, \ldots, w_n)$$

  이를 바탕으로 n-1개의 단어가 나열된 상태에서 n번째 단어의 확률은 다음과 같이 쓸 수 있다.

$$P(w_n|w_1, \ldots, w_{n-1})$$

  전체 단어 시퀀스 W의 확률은 모든 단어가 예측되어야 알 수 있으므로 단어 시퀀스의 확률은 다음과 같다.

$$P(W) = P(w_1, w_2, w_3, w_4, w_5, \ldots w_n) = \prod_{i=1}^{n} P(w_i|w_1, \ldots, w_{i-1})$$

Ex. $P(W) = P(w_1, w_2, w_3) = \prod_{i=1}^{3} P(w_i|w_1, \cdots, w_{i-1}) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)$

→ 문장의 확률은 각 단어들이 이전 단어가 주어졌을 때 다음 단어로 등장할 확률의 곱으로 구성된다.

통계적 언어 모델(Statistical Language Model, SLM)

- 통계적 언어모델은 언어모델의 전통적인 접근 방법

- 문장에 대한 확률
  조건부 확률을 이용해서 문장 "An adorable little boy is spreading smiles"의 확률을 식으로 표현해보자.

$$P(\text{An adorable little boy is spreading smiles}) =$$
$$P(\text{An}) \times P(\text{adorable}|\text{An}) \times P(\text{little}|\text{An adorable}) \times P(\text{boy}|\text{An adorable little}) \times P(\text{is}|\text{An adorable little boy})$$
$$\times P(\text{spreading}|\text{An adorable little boy is}) \times P(\text{smiles}|\text{An adorable little boy is spreading})$$

→ 문장의 확률을 구하기 위해 각 단어에 대한 예측 확률들을 곱해준다.

이 때, 각각의 확률을 카운트에 기반하여 계산한다.

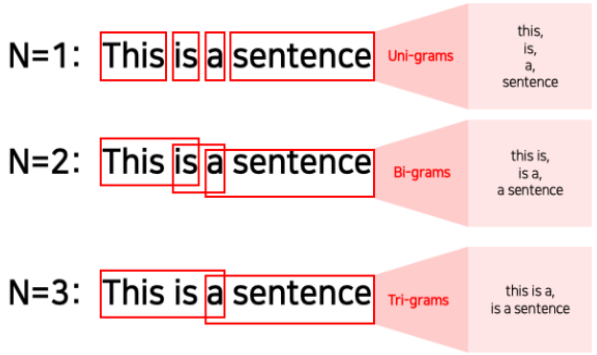
$$P(\text{is}|\text{An adorable little boy}) = \frac{\text{count}(\text{An adorable little boy is})}{\text{count}(\text{An adorable little boy })}$$

희소 문제를 완화시켜보자!

- 데이터에 "An adorable little boy is"가 없다면? → 분자가 0이 되어 해당 단어 시퀀스의 확률은 0이 된다.
- 데이터에 "An adorable little boy"가 없다면? → 분모가 0이 되어 해당 단어 시퀀스 확률은 정의되지 않는다.

희소 문제(Sparsity Problem) 발생

N-gram 언어 모델(N-gram Language Model)



- 여전히 카운트에 기반한 통계적 접근을 사용하므로 SLM의 일종이다.
- 하지만 SLM과 달리 이전에 등장한 모든 단어를 고려하는 것이 아니라 일부 단어만 고려하는 방법을 사용한다.
- N-gram은 N개의 연속적인 단어 나열을 의미하며, N은 이전 단어를 몇 개 볼지(N-1)를 결정한다.
- 갖고 있는 코퍼스에서 n개의 단어 뭉치 단위로 끊어서 이를 하나의 토큰으로 간주한다.

Ex. 4-gram & spreading의 다음 단어 예측

~~An adorable little~~ boy is spreading ?

무시됨!

n-1개의 단어

$$P(w|\text{boy is spreading}) = \frac{\text{count(boy is spreading } w)}{\text{count(boy is spreading)}}$$

if 갖고 있는 코퍼스에서
"boy is spreading"이 1000번 등장
"boy is spreading insults"이 500번 등장
"boy is spreading smiles"이 200번 등장한 경우,

$$P(\text{insults}|\text{boy is spreading}) = 0.500$$

$$P(\text{smiles}|\text{boy is spreading}) = 0.200$$

→ 확률적 선택에 따라 우리는 "insults"가 더 맞다고 판단하게 된다.

N-gram 언어 모델(N-gram Language Model)

[장점]

- Sparsity Problem(희소 문제) 일부 해소
  - N이 작아질수록 해당 단어의 시퀀스를 카운트할 확률을 높일 수 있음

$$P(\text{is}|\text{An adorable little boy}) \approx P(\text{is}|\text{little boy})$$

[단점]

- Sparsity Problem(희소 문제) 여전히 존재
  - 물론 모든 이전 단어들을 보는 것보다 일부 단어만 보는 것이 현실적으로 카운트할 수 있는 확률을 높이지만, 그럼에도 불구하고 여전히 n-gram에 대한 희소 문제가 존재한다.
- N을 선택하는 것은 trade-off 문제
  - N이 커질수록 sparsity problem 문제가 심각해지고, N이 작아질수록 현실의 확률 분포와 멀어진다.

  → 적절한 N을 선택하는 것이 중요하다. (정확도를 높이려면 N은 최대 5를 넘게 잡아서는 안된다고 권장)

→ N-gram 언어 모델의 한계점을 극복하기 위한 Smoothing(스무싱), Backoff(백오프) 등의 여러 일반화 방법들이 존재하지만, 본질적인 취약점을 완전히 해결하지 못하였다. 이를 위한 대안으로 인공 신경망을 이용한 언어 모델이 많이 사용된다.

Perplexity(PPL)

- 언어 모델을 평가하기 위한 평가 지표
- "perplexed"는 "헷갈리는"과 유사한 의미를 가지므로, 여기서 PPL은 "헷갈리는 정도"로 받아들일 수 있다.
- PPL은 낮을수록 언어 모델의 성능이 높다는 것을 의미한다.
- PPL은 문장의 길이로 정규화된 문장 확률의 역수로, 문장 W의 길이를 N이라 할때 다음과 같다.

$$PPL(W) = P(w_1, w_2, w_3, \ldots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \ldots, w_N)}} = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i|w_1, w_2, \ldots, w_{i-1})}}$$

- N-gram을 적용할 경우(bi-gram인 경우)

$$PPL(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i|w_{i-1})}}$$

- N-gram에서 N값이 커질수록 PPL값은 작아지게 되며, 더 높은 확률을 보여주는 경향을 보인다

$$\text{if } m \geq n$$

$$P(w_n|w_1, \ldots, w_{n-1}) \leq P(w_m|w_1, \ldots, w_{m-1})$$

	Uni-gram	Bi-gram	Tri-gram
PPL	962	170	109

같은 corpus로 학습한 N-gram 모델에서 n의 값에 따라 PPL값이 바뀌는것을 확인

Perplexity(PPL)

- PPL은 선택할 수 있는 가능한 경우의 수를 의미하는 분기 계수(branching factor)다.
- PPL은 이 언어 모델이 특정 시점에 평균적으로 몇 개의 선택지를 가지고 고민하는지를 의미한다.

$$PPL(W) = \underline{P(w_1, w_2, w_3, \ldots, w_N)^{-\frac{1}{N}}} = \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} = \frac{1}{10}^{-1} = 10$$

$$= \prod_{i=1}^{N} P(w_i | w_1, \cdots, w_{i-1}) = P(w_1)P(w_2|w_1)P(w_3|w_1,w_2)\cdots$$

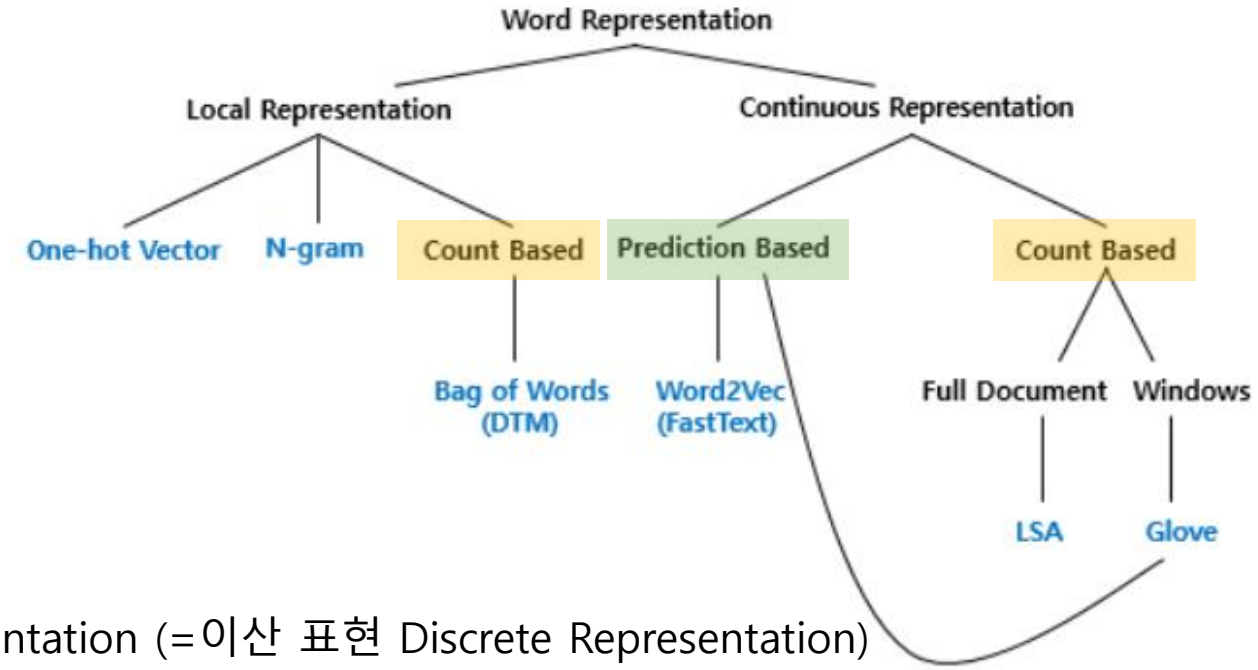
→ 해당 언어 모델은 테스트 데이터에 대해서 다음 단어를 예측하는 모든 시점마다 평균 10개의 단어를 가지고 어떤 것이 정답인지 고민하고 있다고 볼 수 있다.

- 하지만 PPL 값이 낮다는 것은 테스트 데이터 상에서 높은 정확도를 보인다는 것이지, 사람이 보기에 좋은 언어 모델이라는 것을 의미하지는 않는다.
- 언어 모델의 PPL은 테스트 데이터에 의존하므로 2개 이상의 언어 모델을 비교할 때는 정량적으로 양이 많고, 도메인에 알맞은 동일한 테스트 데이터를 사용해야 신뢰도가 높다.

Contents

Unit 04 | 단어 의미 & 단어 표현

어떻게 단어를 "표현"할 것인가?

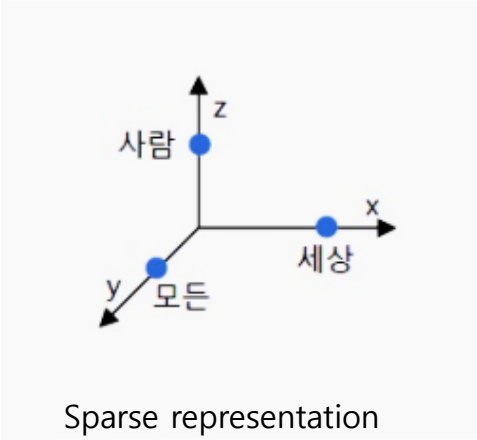


- 국소 표현 Local Representation (=이산 표현 Discrete Representation)
  : 해당 단어 자체만 보고, 특정 값을 매핑하여 단어를 표현하는 방법, 단어의 뉘앙스 표현 불가능
- 분산 표현 Distributed Representation (=연속 표현 Continuous Representation)
  : 주변 단어를 참고하여 단어를 표현하는 방법, 단어의 뉘앙스 표현 가능

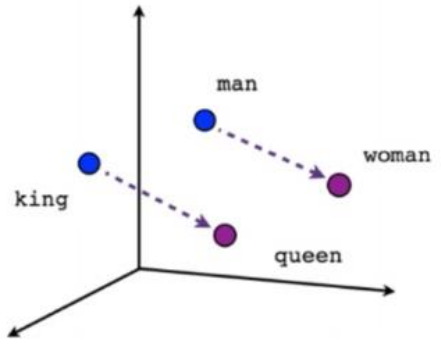
Unit 04 | 단어 의미 & 단어 표현

어떻게 단어를 "표현"할 것인가?

- 희소 표현(Sparse Representation)
  - 벡터 또는 행렬의 값이 대부분 0으로 표현되는 방법
  - 각 단어 벡터 간 유의미한 유사성을 표현할 수 없음
  - Ex. One-hot vector [0,0,0,1,0,...,0]



Sparse representation



Distributed representation

- 분산 표현(Distributed Representation)
  - 단어의 의미를 다차원 공간에 벡터화하는 방법
  - 분포 가설 가정 하에 만들어진 표현 방법
    - 비슷한 문맥에서 등장하는 단어들은 비슷한 의미를 가진다.
  - 벡터의 차원이 단어 집합 크기일 필요 없음
    → 상대적으로 저차원
  - 단어 벡터 간 유의미한 유사도 계산 가능

→ 분산 표현을 이용하여 단어 간 의미적 유사성을 벡터화하는 작업

: Word Embedding(워드 임베딩)

Unit 04 | 단어 의미 & 단어 표현

어떻게 의미를 "벡터"에 넣을 것인가?

- [Ans] 자연어의 통계적 패턴 정보를 벡터에 넣자!
  - [Why] 자연어의 의미는 해당 언어 화자들의 사용에서 드러나기 때문이다.

	BoW 가정	언어 모델	분포 가설
내용	빈도	순서	맥락
대표 통계	TF-IDF	-	PMI
대표 모델	-	GPT	Word2Vec

통계 기반 ← → 신경망 기반

- BoW 가정 : 순서 X 빈도 O
  → 저자의 의도는 단어 사용 여부 혹은 빈도에서 드러난다.
- Language model : 순서 O
  → 주어진 단어 시퀀스가 얼마나 자연스러운지 정도를 확인해야 한다.
- 분포 가설 : 주변 단어
  → 단어의 의미는 주변 context를 통해 유추 가능하다.

Unit 04 | 단어 의미 & 단어 표현

Bags of Words (BoW)

- Bag of Words란 단어들의 순서는 전혀 고려하지 않고,
  단어들의 출현 빈도(frequency)에만 집중하는 텍스트 데이터의 수치화 표현 방법

  - BoW 만드는 과정

    (1) 각 단어에 고유한 정수 인덱스를 부여합니다. # 단어 집합 생성.

    (2) 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터를 만듭니다.

  - BoW는 각 단어가 등장한 횟수를 수치화하는 텍스트 표현 방법이므로 주로 어떤 단어가 얼마나 등장했는지를 기준으로 문서가 어떤 성격의 문서인지를 판단하는 작업에 쓰입니다.
    즉, 분류 문제나 여러 문서 간의 유사도를 구하는 문제에 주로 쓰입니다.
  - Ex) '달리기', '체력', '근력'과 같은 단어가 자주 등장 -> 체육 관련 문서로 분류
  - '미분', '방정식', '부등식'과 같은 단어가 자주 등장 -> 수학 관련 문서로 분류

Unit 04 | 단어 의미 & 단어 표현

문서 단어 행렬(Document-Term Matrix, DTM)

- BoW와 다른 표현 방법이 아니라 BoW 표현을 다수의 문서에 대해서 행렬로 표현하고 부르는 용어
- 다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것

Ex. 문서 1 : 먹고 싶은 사과
문서 2 : 먹고 싶은 바나나
문서 3 : 길고 노란 바나나 바나나
문서 4 : 저는 과일이 좋아요

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

Unit 04 | 단어 의미 & 단어 표현

문서 단어 행렬(Document-Term Matrix, DTM)

[단점]

- 희소 표현(Sparse Representation)
  - DTM도 one-hot 벡터와 마찬가지로 대부분의 값이 0이므로 공간 낭비 + 계산량 증가
  - DTM에서도 각 행을 문서 벡터라고 할 경우, 각 문서 벡터의 차원 == 전체 단어 집합 크기
  - 물론 구두점, 빈도수가 낮은 단어, 어간이나 표제어 추출을 통해 단어를 정규화해 단어 집합의 크기를 어느 정도 줄일 수는 있음.
- 단순 빈도 수 기반 접근
  - 여러 문서에 등장하는 모든 단어에 대해 빈도를 표기하는 방법은 한계를 가짐
    ex. 'the'의 경우, 어느 문서에서나 등장함. 하지만 문서의 'the' 빈도수가 높다고 해서 유사하다고 판단할 수 없음
  - 각 문서에는 중요한 단어와 불필요한 단어가 혼재되어 있음

    중요한 단어에 대해 가중치를 주자! → TF-IDF

TF-IDF(Term Frequency-Inverse Document Frequency)

- 단어의 빈도와 역 문서 빈도를 사용해 DTM 내의 각 단어들마다 중요한 정도를 가중치로 표현한다.
- 우선 DTM을 만든 후, TF-IDF 가중치를 부여해 조정한다.

[TF, DF, IDF 정의]

- **tf(d,t)** : 특정 문서 d에서 특정 단어 t의 등장 횟수 (DTM에서의 값)
- **df(t)** : 특정 단어 t가 등장한 문서의 수
  → 특정 단어가 등장한 문서 수에만 관심을 가짐. (몇 번 등장했는지는 관심 X)
  → ex. 바나나의 df는 2 (문서2, 문서3에서 등장)
- **idf(d,t)** : df(t)에 반비례 하는 수 (n은 문서의 총 개수)

$$idf(d,t) = log(\frac{n}{1+df(t)})$$

총 문서의 수가 커질수록, IDF의 값이 기하급수적으로 커지는 것을 방지

특정 단어가 전체 문서에서 등장하지 않을 경우, 분모가 0이 되는 상황을 방지

Unit 04 | 단어 의미 & 단어 표현

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 TF와 IDF를 곱한 값을 의미한다.

$$idf(d,t) = log(\frac{n}{1+df(t)})$$

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

DTM
(=TF)

×

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

IDF

=

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

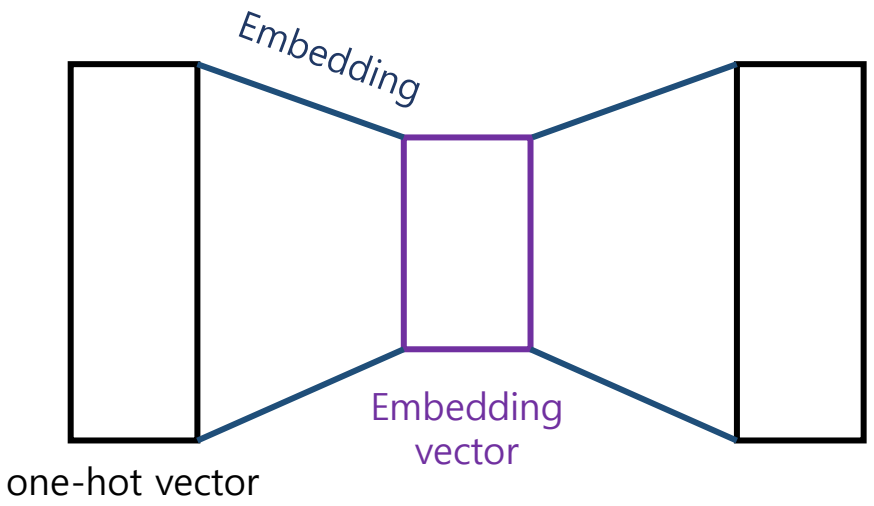
TF-IDF

* 문서 2의 바나나와 문서 3의 바나나의 TF-IDF 가중치가 다른 이유
: TF-IDF는 특정 문서에서 자주 등장하는 단어를 그 문서 내에서 중요한 단어로 판단하기 때문에 바나나가 1번 언급된 문서 2보다 2번 언급된 문서 3에서 바나나를 더욱 중요한 단어라고 판단하는 것이다.

Contents

워드 임베딩(Word Embedding)

- 각 단어를 인공 신경망 학습을 통해 벡터화하는 워드 임베딩이라는 방법
- 단어를 벡터로 표현하는 방법으로, 단어를 밀집 표현으로 변환

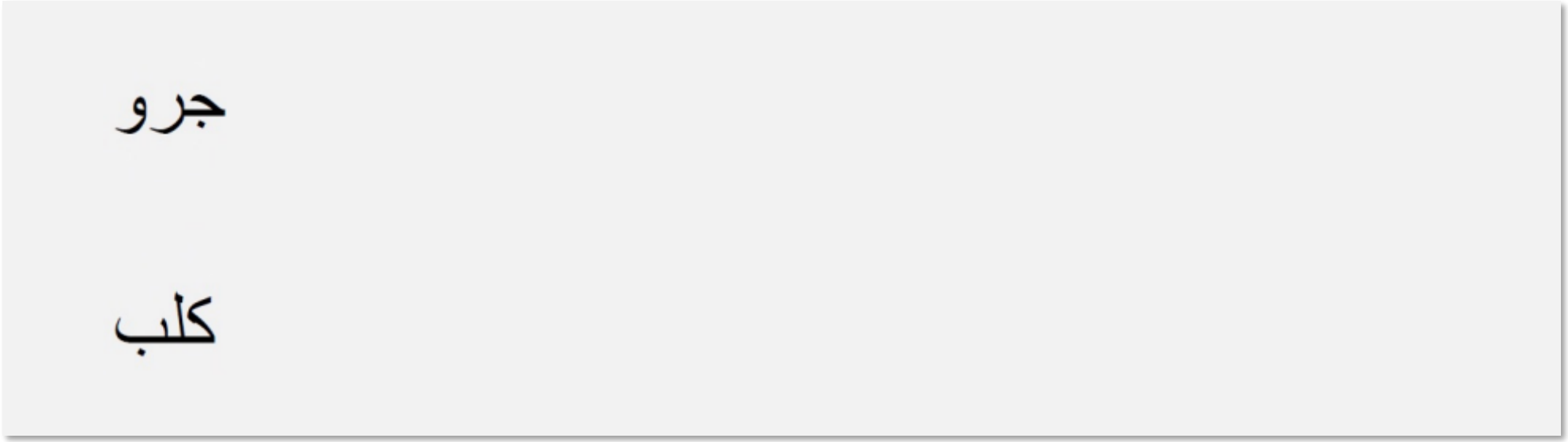


- **이때, 무슨 정보를 이용해 단어 의미를 표현?**

주변단어	→	Word2Vec
주변단어 + n-gram	→	FastText
주변단어 + 분포	→	GloVe

Word2Vec

- 한 단어의 주변 단어를 통해, 그 단어의 의미를 파악



컴퓨터에게 자연어는 "기호"일 뿐

Word2Vec

- 한 단어의 주변 단어를 통해, 그 단어의 의미를 파악

جرو (개) 가 멍멍! 하고 짖었다.

كلب (강아지) 가 멍멍! 하고 짖었다.

각 단어의 의미를 모르겠지만, 주변 단어가 비슷하니 의미도 비슷할 것이다!

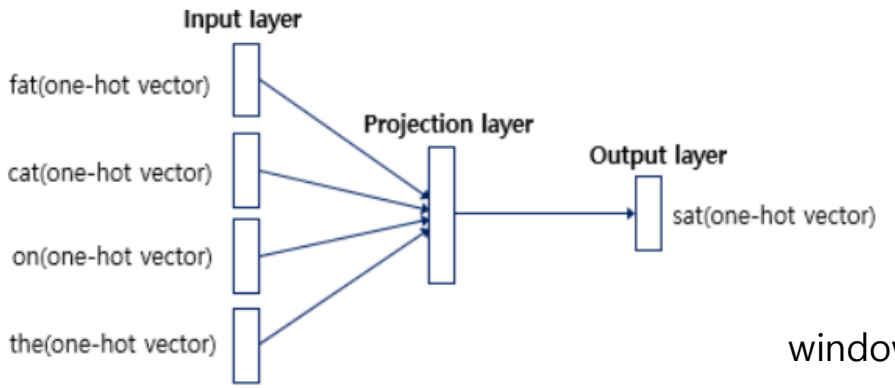
Word2Vec

- CBOW 방식과 Skip-gram 방식으로 나뉨

CBOW

The fat cat ___ on the mat

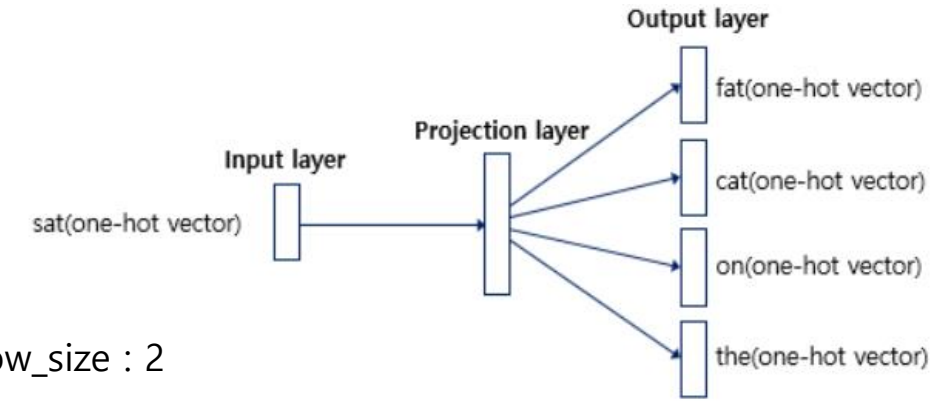
주변에 있는 단어들을 가지고
중간에 있는 단어를 예측하는 방법



Skip-gram

The ___ ___ sat ___ ___ mat

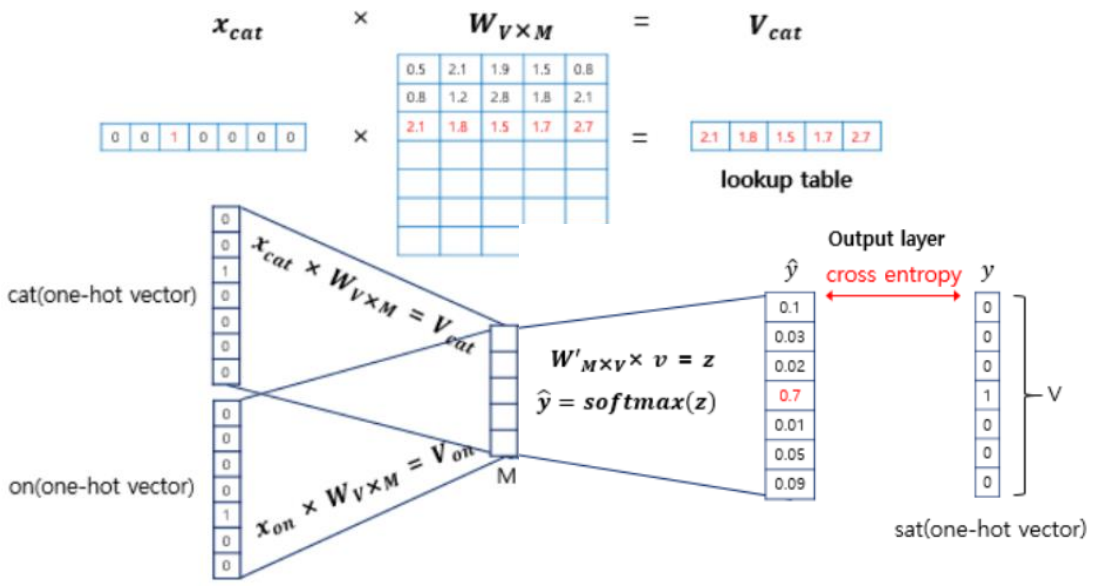
중앙의 단어로부터 주변의 여러 단어를
예측하는 방법



window_size : 2

Word2Vec - CBOW

- 주변 단어들로부터 중심 단어를 예측



"The fat cat sat on the mat"

- window = 1이라 가정
- 중간 단어를 예측하기 위해 주변 단어 벡터를 평균 냄
- W와 W′는 다른 행렬 (전치 X)
- W가 결국 임베딩 벡터가 됨

input vector : one-hot vector

hidden vector : input vector * W_{VM}

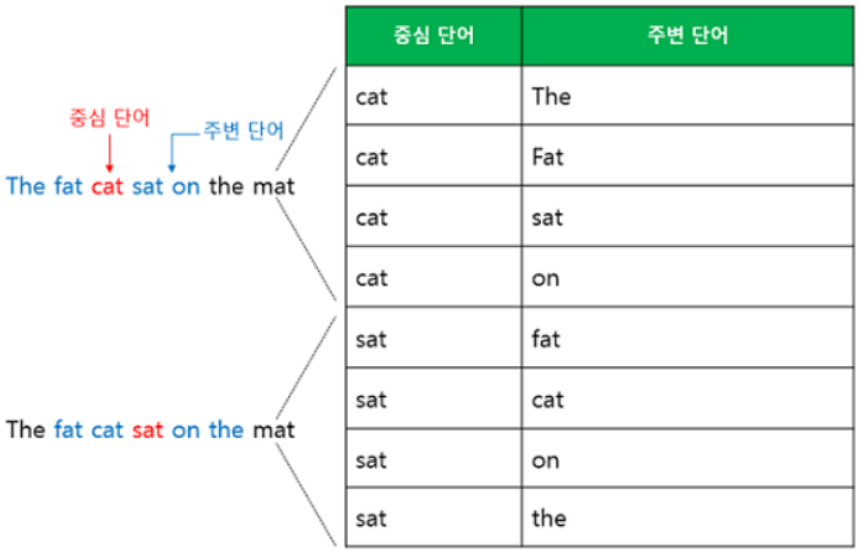
→ 일종의 lookup table로 계산해서 나온 저차원의 밀집 벡터를 바탕으로 다시 W′ 행렬을 곱해서 output 벡터를 구함 → softmax로 확률 값이 가장 큰 단어가 중심 단어가 됨 → 중심단어로 예측되는 확률값과 타겟 (정답)값과의 차이를 비교해서 cross entropy를 통해 학습함

V : 단어 집합 크기

M : 히든 레이어의 차원 크기

Word2Vec – Skip-gram

- 중심 단어로부터 주변 단어를 예측
- CBOW의 반대 개념



중심 단어	주변 단어
cat	The
cat	Fat
cat	sat
cat	on
sat	fat
sat	cat
sat	on
sat	the

CBOW

Input	Output
fat, cat	The
The, cat, sat	fat
The, fat, sat, on	cat
fat, cat, on, the	sat
cat, sat, the, mat	on
sat, on, mat	the
on, the	mat

word	count
The	1
fat	1
cat	1
sat	1
on	1
the	1
mat	1

Skip-gram

Input	Output
The	fat, cat
fat	The, cat, sat
cat	The, fat, sat, on
sat	fat, cat, on, the
on	cat, sat, the, mat
the	sat, on, mat
mat	on, the

word	count
The	2
fat	3
cat	4
sat	4
on	4
the	3
mat	2

주변 단어로부터 오직 1개의 타겟 단어를 예측 및 학습. 타겟 단어를 바탕으로 여러 문맥 단어를 예측 및 학습.

- 보통 Skip-gram이 CBOW보다 성능이 좋음

[Why]

- 역전파 시, 더 많은 정보를 받음
- 타겟 단어 1개 당 더 많은 학습 데이터를 가지고 있음 (여러 문맥에 걸쳐 단어를 학습)

Word2Vec

[장점]

- 단어 간 유사도 측정 가능, 관계 파악 가능
- 벡터 연산을 통해 추론 가능

[단점]

- 단어의 Subword Information 무시

  Ex. "birthplace(출생지)"

  → "birth"와 "place"만 알면 무슨 뜻인지 유추 가능 Subword Information 문제를 보완하자! → FastText

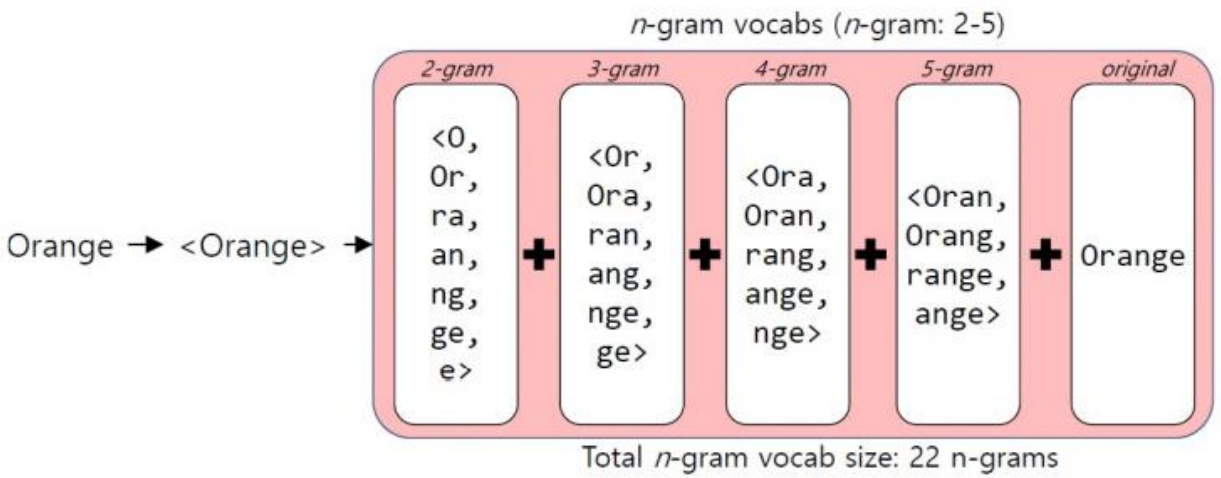
  → 하지만, Word2Vec에서는 한 단어를 쪼개지 않고 따로 따로 인식하기 때문에 이를 무시한다고 할 수 있음
- Out of Vocabulary (OOV)
  - 코퍼스를 통해서 학습이 되지 않은 단어는 데이터가 없으므로 OOV 문제 발생

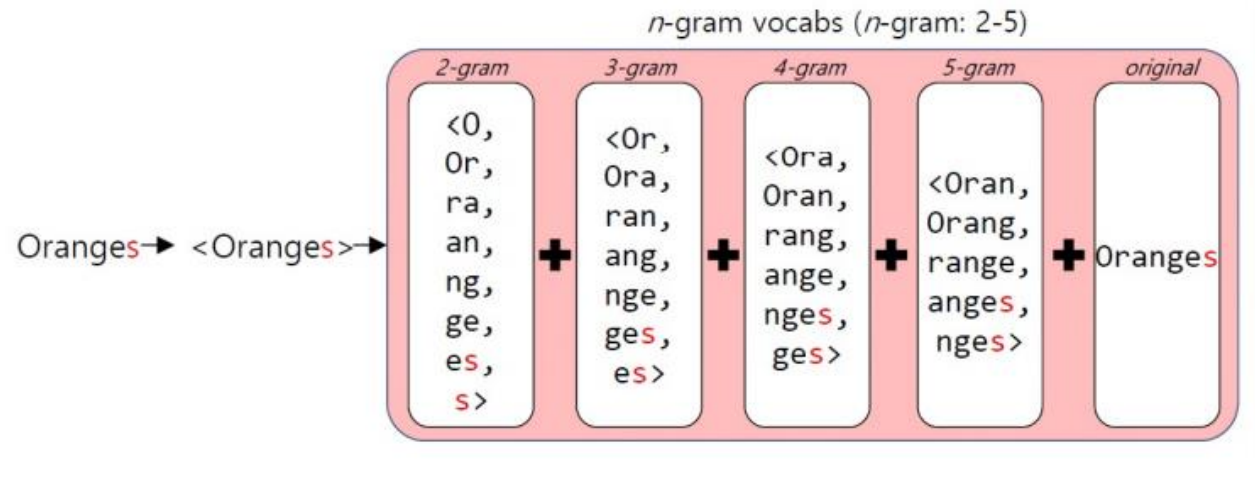
FastText

하지만, 주변 단어만 반영해도 괜찮을까?

전체 코퍼스 통계 정보도 고려해보자! → GloVe

- Word2Vec과 유사한 방식으로 학습
- [차이] 단어를 n-gram으로 나누어 학습 (subword를 고려=하나의 단어 안에도 여러 단어들이 존재하는 것으로 간주)
- 이 때, n-gram으로 나누어진 단어는 사전에 들어가지 않고, 별도의 n-gram 벡터 생성
- [장점] OOV, rare word에 대해 대응 가능





GloVe

[기존 연구의 한계]

- TF-IDF의 경우, 카운트 기반으로 전체 통계 정보를 고려하지만, 단어의 의미 유추 작업의 성능은 낮음
- Word2Vec의 경우, 예측 기반으로 단어 간 유추 작업엔 상대적으로 뛰어나지만, 코퍼스의 전체적인 통계 정보를 반영하지 못함

→ 카운트 기반과 예측 기반 방법론을 모두 사용하자!

[윈도우 기반 동시 등장 행렬]

- 단어의 동시 등장 행렬은 행과 열을 전체 단어 집합의 단어들로 구성하고, i 단어의 윈도우 크기 내에서 k 단어가 등장한 횟수를 i행 k열에 기재한 행렬이다.

- I like deep learning
- I like NLP
- I enjoy flying
- N은 1이라 가정

카운트	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	0	0	1
deep	0	1	0	0	1	0	0
learning	0	0	0	1	0	0	0
NLP	0	1	0	0	0	0	0
flying	0	0	1	0	0	0	0

GloVe

[동시 등장 확률(Co-occurrence Probability)]

- P(k|i)는 동시 등장 행렬로부터 특정 단어 i의 전체 등장 횟수를 카운트하고, 특정 단어 i가 등장했을 때 어떤 단어 k가 등장한 횟수를 카운트하여 계산한 조건부 확률
- i를 중심 단어, k를 주변 단어라 할 때, 앞에서 배운 동시 등장 행렬에 중심 단어 i행의 모든 값을 더한 것이 분모가 되고, i행 k열의 값이 분자가 된다.

동시 등장 확률과 크기 관계 비(ratio)	k=solid	k=gas	k=water	k=fasion
P(k | ice)	0.00019	0.000066	0.003	0.000017
P(k | steam)	0.000022	0.00078	0.0022	0.000018
P(k | ice) / P(k | steam)	8.9	0.085	1.36	0.96

- P(solid | ice) > P(solid | steam) → 비율 8.9로 큼
- P(gas | ice) < P(gas | steam) → 비율 0.085로 작음
- Ice와 steam과 모두 관계가 있는 water의 경우 확률이 비슷 → 비율 1에 가까움
- Ice와 steam과 모두 관계가 없는 fashion의 경우 확률이 비슷 → 비율 1에 가까움

→ 이런 특징을 바탕으로 glove 식이 설계가 됨

GloVe

- Word2Vec이 전체 코퍼스의 정보를 담지 못한다는 문제 보완
- 두 단어의 유사도에 통계 정보가 반영됨
- 분류 → 회귀 문제로 전환
- [목적 함수]

$$dot\ product(w_i\ \tilde{w}_k) \approx\ log\ P(k \mid i) = log\ P_{ik}$$

'임베딩된 중심 단어와 주변 단어 벡터의 내적이 전체 코퍼스에서의 동시 등장 확률이 되도록 만드는 것'

$$Loss\ function = \sum_{m,n=1}^{V} f(X_{mn})(w_m^T \tilde{w}_n + b_m + \tilde{b_n} - logX_{mn})^2$$

$f(x) = min(1,\ (x/x_{max})^{3/4})$ → 특정 단어가 지나치게 빈도수가 높아서 X_{mn}이 튀는 현상을 방지하기 위해 추가한 함수

- X : 동시 등장 행렬(Co-occurrence Matrix)
- X_{ij} : 중심 단어 i가 등장했을 때 윈도우 내 주변 단어 j가 등장하는 횟수
- X_i : $\sum_j X_{ij}$: 동시 등장 행렬에서 i행의 값을 모두 더한 값
- P_{ik} : $P(k \mid i)$ = $\frac{X_{ik}}{X_i}$: 중심 단어 i가 등장했을 때 윈도우 내 주변 단어 k가 등장할 확률
  Ex) P(solid I ice) = 단어 ice가 등장했을 때 단어 solid가 등장할 확률
- $\frac{P_{ik}}{P_{jk}}$: P_{ik}를 P_{jk}로 나눠준 값
  Ex) P(solid I ice) / P(solid I steam) = 8.9
- w_i : 중심 단어 i의 임베딩 벡터
- $\tilde{w}_k$: 주변 단어 k의 임베딩 벡터

GloVe

$$dot\ product(w_i\ \tilde{w}_k) \approx\ log\ P(k \mid i) = log\ P_{ik}$$

$$F(w_i,\ w_j,\ \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

두 단어 사이의 비율을 encode하기 위해 벡터 차를 입력값으로!

$$F(w_i - w_j,\ \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{ik}}$$

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

$$F(w_i^T \tilde{w}_k - w_j^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

조건을 만족하는 F는 지수 함수이므로 다음과 같이 식을 바꾸면,

$$exp(w_i^T \tilde{w}_k - w_j^T \tilde{w}_k) = \frac{exp(w_i^T \tilde{w}_k)}{exp(w_j^T \tilde{w}_k)}$$

$$exp(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

$$w_i^T \tilde{w}_k = log\ P_{ik} = log\ (\frac{X_{ik}}{X_i}) = log\ X_{ik} - log\ X_i$$

이 때, w_i와 w_k의 내적은 순서가 바뀌어도 값이 같도록 일종의 편향 b를 추가해 표현

$$w_i^T \tilde{w}_k + b_i + \tilde{b_k} = log\ X_{ik}$$

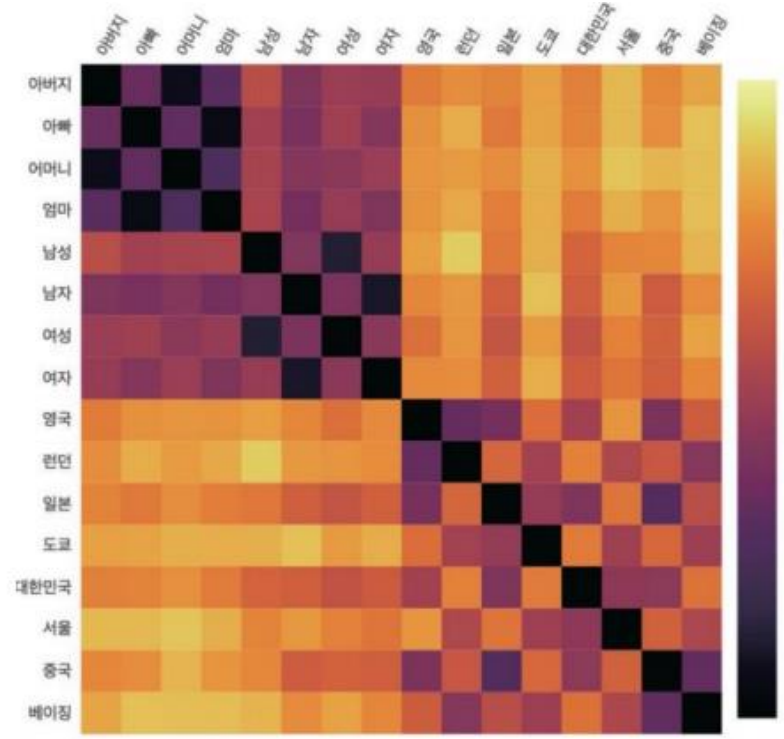
하지만, 주변 단어 및 코퍼스 통계 정보만 고려하는게 맞나?

문맥에 맞춰서 임베딩 하자! → Contextualized Word Embedding (ELMo, BERT ...)

임베딩으로 할 수 있는 것

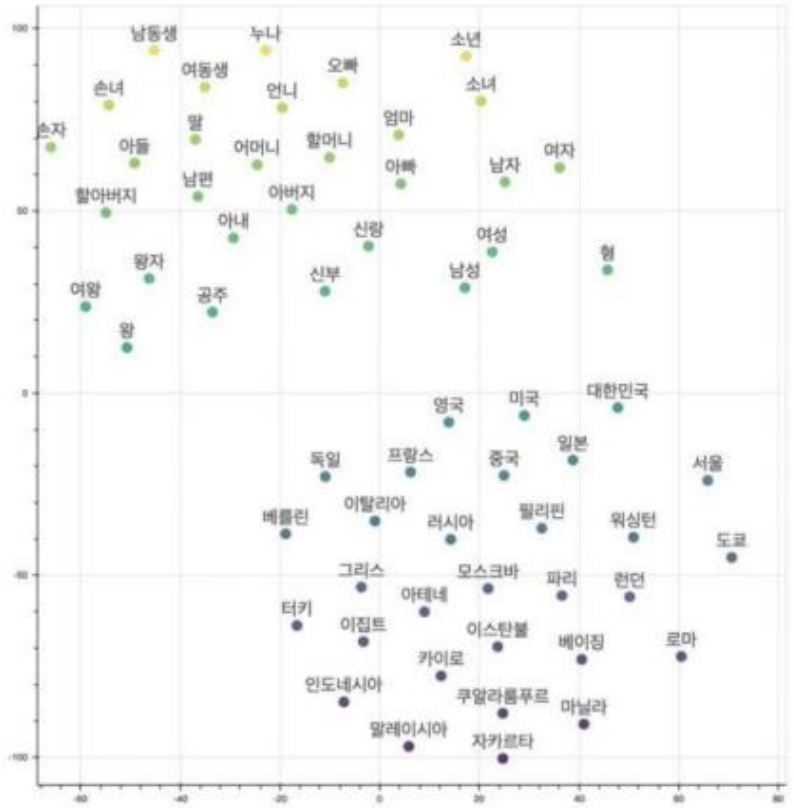
[관련도/유사도 계산]

희망	절망	학교	학생	가족	자동차
소망	체념	초등	대학생	아이	승용차
행복	고뇌	중학교	대학원생	부모	상용차
희망찬	절망감	고등학교	고학생	편부모	트럭
꿈	상실감	야학교	교직원	고달픈	대형트럭
열망	번민	중학	학부모	사랑	모터사이클



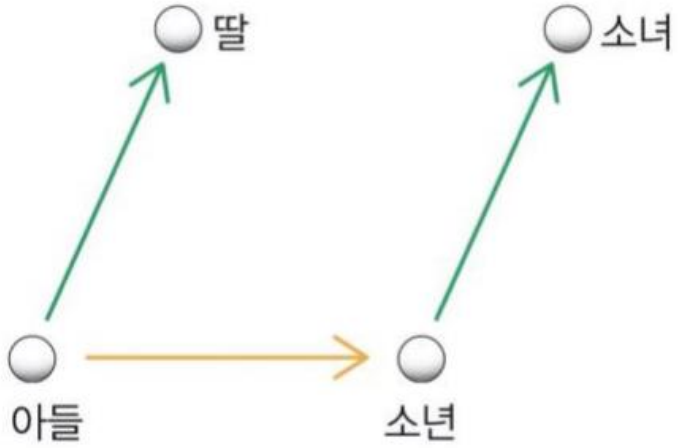
임베딩으로 할 수 있는 것

[시각화]



임베딩으로 할 수 있는 것

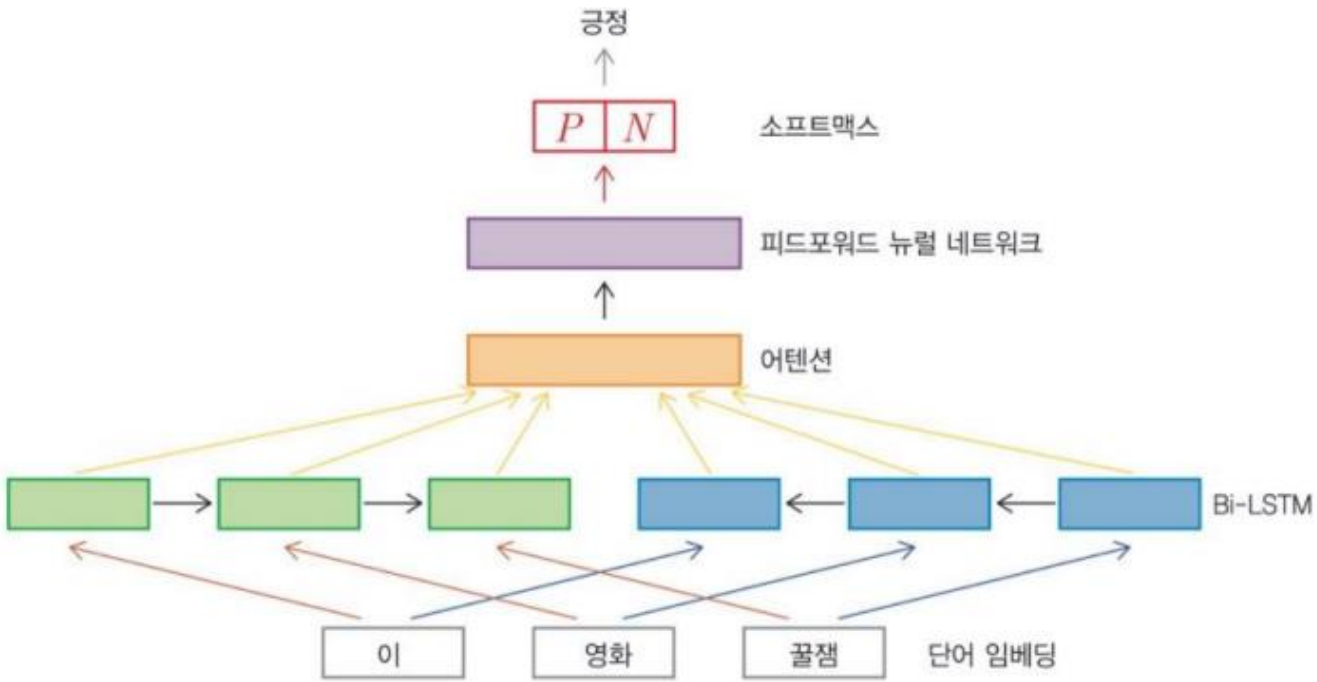
[벡터 연산(유추 평가) : 아들 – 딸 + 소녀 = 소년]



단어1	단어2	단어3	결과
아들	딸	소년	소녀
아들	딸	아빠	엄마
아들	딸	남성	여성
남동생	여동생	소년	소녀
남동생	여동생	아빠	엄마
남동생	여동생	남성	여성
신랑	신부	왕	여왕
신랑	신부	손자	손녀
신랑	신부	아빠	엄마

임베딩으로 할 수 있는 것

[전이학습(Transfer Learning) : 다른 모델의 입력값]



Homework

과제 : NLP 맛보기

STEP 1. 데이터 확인

STEP 2. Tokenizing (불용어 처리, 특수 문자 제거 등의 전처리 포함)

STEP 3. 임베딩 (One-hot encoding, CBOW, Skip-gram, GloVe, FastText 등)

STEP 4. 유의미한 해석 도출 (유사도, wordcloud, 이진 분류 모델, 그래프 해석 …)

[**주의사항**]

- 임베딩 모델 적어도 2개 이상 사용 후, 해석에 따라 가장 좋은 모델을 선택
- 유의미한 해석 도출해보기 – 3가지 이상의 인사이트 도출
- 토크나이저 및 임베딩 모델 선택 과정, 인사이트 해석을 주석으로 달아주세요!

참고자료

- ToBig's 18기 정규세션 NLP Basic 강의 (임수진님)
- https://wikidocs.net/book/2155
- https://heeya-stupidbutstudying.tistory.com/56
- https://judia.tistory.com/2
- https://heytech.tistory.com/353
- https://hong-yp-ml-records.tistory.com/57
- https://wikidocs.net/22885
- 한국어를 위한 어휘 임베딩의 개발 -1- (brunch.co.kr)
- 자연어처리 바이블 (임희석 | 고려대학교 자연어처리연구실 저)

감사합니다