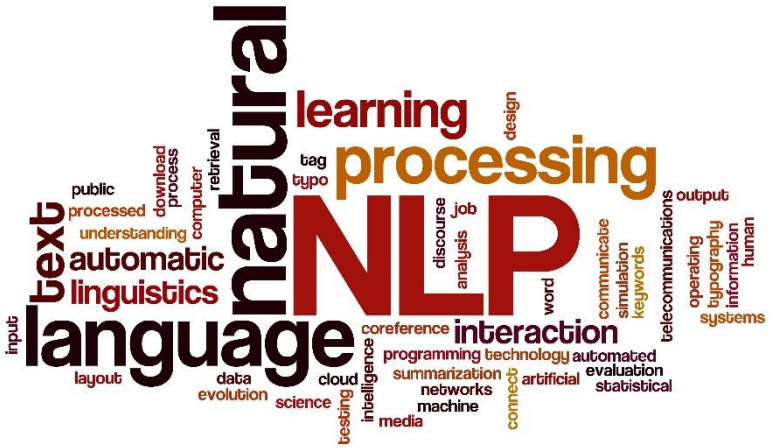
빅데이터 분석
자연어처리(Natural Language Process)

Jaekwan Ahn

한국산업기술대학교
KOREA POLYTECHNIC UNIVERSITY

자연어 처리(NLP)란?

- 자연어(natural language)란 우리가 일상 생활에서 사용하는 언어를 말한다. 자연어 처리(natural language processing)란 이러한 자연어의 의미를 분석하여 컴퓨터가 처리할 수 있도록 하는 일을 일컫는다.
  - 자연어 처리는 음성 인식, 내용 요약, 번역, 사용자의 감성 분석, 텍스트 분류 작업(스팸 메일 분류, 뉴스 기사 카테고리 분류), 질의 응답 시스템, 챗봇과 같은 곳에서 사용되는 분야이다.
  - 최근 딥 러닝이 주목을 받으면서, 인공지능이 IT 분야에서 NLP가 중요 키워드로 떠오르고 있다. 자연어 처리는 기계에게 인간의 언어를 이해시킨다는 점에서 인공지능에 있어서 가장 중요한 연구 분야이면서도, 아직도 정복되어야 할 산이 많은 분야이다.



자연어 처리(NLP) 최신 트랜드

- Generative Pre-trained Transformer3 (GPT3)는 강력한 언어모델로 4990억 개 데이터 셋 중 가중치를 샘플링해서 3000억개로 구성된 데이터셋으로 사전 학습 한 모델
- 훈련 비용만 4천 600만 달러 정도가 소요되는 것으로 추정, 이는 한화로 50억원. 처리는 음성 인식, 내용 요약, 번역, 사용자의 감성 분석, 텍스트 분류 작업(스팸 메일 분류, 뉴스 기사 카테고리 분류), 질의 응답 시스템, 챗봇과 같은 곳에서 사용되는 분야이다.
- https://twitter.com/sharifshameem/status/1282676454690451457

GPT-3, 인류 역사상 가장 뛰어난 '언어 인공지능'이다

김종윤 스캐터랩 대표 | 승인 2020.08.14 21:10

> AI... 더 높고 과감한 목표를 현실적으로 가지게 되었다는 게 GPT-3의 가장 큰 의미로 보여진다. AI의 발전은 우리의 예상보다 빠른 속도로 진행되고 있다. 바로 그런 시기에 그 영역의 일을 하고 있다는 게 행운으로 여겨지는 요즘이다.

인공지능(AI) 자연어 처리(NLP)에서 가장 화제가 되고 있는 플랫폼으로는 구글의 양방향 언어모델 버트(Bert), OpenAI의 단방향 언어모델 GPT-2, 기계신경망 번역(Transformer) 모델 등을 꼽을 수 있다.

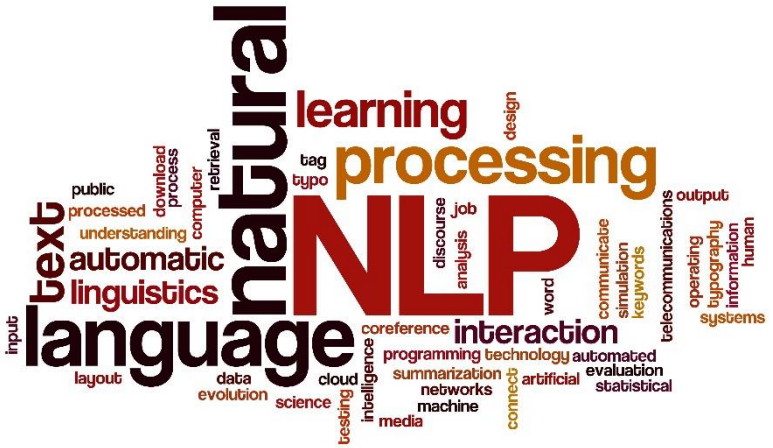
여기에 지난 6월 1일(현지시간) OpenAI가 새로운 강력한 언어 모델 'GPT-3(Generative Pre-Training 3)'를 아카이브(arXiv)를 통해 공개했다. 이는 지난해 초에 공개한 소설 쓰는 인공지능 'GPT-2' 보다 훨씬 더 크고 혁신적인 버전으로 진화된 모델이다.

이 모델은 4990억개 데이터셋 중에서 가중치 샘플링해서 3000억(300B)개로 구성된 데이터셋으로 사전 학습을 받았으며, 1750억개(175Billion) 매개 변수로 딥러닝의 한계까지 추진돼 미세 조정없이 여러 자연어 처리 벤치마크에서 최첨단 성능을 달성했다. 발표된 내용이라면 단 몇 개 키워드만 넣으면 작문을 작성해주는 혁신적인 AI 언어생성 모델이자 알고리즘인 것이다.

필자는 뛰어난 언어 능력으로 관련분야 최고의 핫 이슈로 떠오르는 이 언어 생성 모델 'GPT-3'가 어떤 능력을 가지고 있고, 어떻게 이런 능력을 갖게 되었는지 알아본다.

Text 전처리

- 자연어 처리에 있어서 텍스트 전처리는 매우 중요한 작업이다. 텍스트 전처리는 용도에 맞게 텍스트를 사전에 처리하는 작업인데, 텍스트에 대해서 제대로 된 전처리를 하지 않으면 자연어 처리 기법들이 제대로 동작하지 않는다.
  - 전처리 과정은 다음과 같다.
  - 토큰화(Tokenization)
  - 정제 및 정규화(Normalization)
  - 어간 추출(Stemming)
  - 표제어 추출(Lemmatization)
  - 불용어(Stopping words)



Text 전처리: 토큰화

- 주어진 코퍼스(corpus)에서 토큰(token)이라 불리는 단위로 나누는 작업을 토큰화(tokenization)라고 부릅니다. 토큰의 단위가 상황에 따라 다르지만, 보통 의미있는 단위로 토큰을 정의한다. 여기서 코퍼스란 한글로 말뭉치라고 표현하는데 자연어 연구를 위해 특정한 목적을 가지고 언어의 표본을 추출한 집합이다.
- 토큰의 기준을 단어(word)로 하는 경우, 단어 토큰화(word tokenization)라고 합니다. 다만, 여기서 단어(word)는 단어 단위 외에도 단어구, 의미를 갖는 문자열로도 간주되기도 합니다.
  - 예시: 입력으로부터 구두점(punctuation)과 같은 문자는 제외시키는 간단한 단어 토큰화 작업을 해보자. 구두점이란, 마침표(.), 컴마(,), 물음표(?), 세미콜론(;), 느낌표(!) 등과 같은 기호를 말합니다.

Input: Time is an illusion. Lunchtime double so!
Output: “Time”, “is”, “an”, “illustion”, “Lunchtime”, “double”, “so”

Text 전처리: 정제와 정규화

- 토큰화 작업 전, 후에는 텍스트 데이터를 용도에 맞게 정제(cleaning) 및 정규화(normalization)하는 일이 항상 함께 한다. 정제 및 정규화의 목적은 각각 다음과 같다.
  - 정제(cleaning) : 갖고 있는 코퍼스로부터 노이즈 데이터를 제거한다.
  - 정규화(normalization) : 표현 방법이 다른 단어들을 통합시켜서 같은 단어로 만들어준다.
- 정제 작업은 토큰화 작업에 방해가 되는 부분들을 배제시키고 토큰화 작업을 수행하기 위해서 토큰화 작업보다 앞서 이루어지기도 하지만, 토큰화 작업 이후에도 여전히 남아있는 노이즈들을 제거하기위해 지속적으로 이루어지기도 한다. 사실 완벽한 정제 작업은 어려운 편이라서, 대부분의 경우 이 정도면 됐다.라는 일종의 합의점을 찾기도 한다.

Text 전처리: 정제와 정규화

- 정제와 정규화 기법은 여러 존재하는데 대표적으로 대,소문자 통합이 있다.
  - 영어권 언어에서 대, 소문자를 통합하는 것은 단어의 개수를 줄일 수 있는 또 다른 정규화 방법이다. 영어권 언어에서 대문자는 문장의 맨 앞 등과 같은 특정 상황에서만 쓰이고, 대부분의 글은 소문자로 작성되기 때문에 대, 소문자 통합 작업은 대부분 대문자를 소문자로 변환하는 소문자 변환작업으로 이루어지게 된다.
  - 소문자 변환이 왜 유용하냐면, 가령, Automobile이라는 단어가 문장의 첫 단어였기때문에 A가 대문자였다고 하면, 여기에 소문자 변환을 사용하면, automobile을 찾는 질의(query)에서, 결과로서 Automobile도 찾을 수 있게 된다.
  - 물론, 대문자와 소문자를 무작정 통합해서는 안 된다. 대문자와 소문자가 구분되어야 하는 경우도 있기 때문! 가령 미국을 뜻하는 단어 US와 우리를 뜻하는 us는 구분되어야 한다.

Text 전처리: 표제어 추출(Lemmatization)

- 표제어(Lemma)는 한글로는 '표제어' 또는 '기본 사전형 단어' 정도의 의미를 갖는데, 표제어 추출은 단어들로부터 표제어를 찾아가는 과정입니다. 표제어 추출은 단어들이 다른 형태를 가지더라도, 그 뿌리 단어를 찾아가서 단어의 개수를 줄일 수 있는지 판단하는 과정이다.
- 예를 들어서 am, are, is는 서로 다른 스펠링이지만 그 뿌리 단어는 be라고 볼 수 있다. 이 때, 이 단어들의 표제어는 be라고 합니다.

Text 전처리: 표제어 추출(Lemmatization)

- 표제어 추출을 하는 가장 섬세한 방법은 단어의 형태학적 파싱을 먼저 진행하는 것이다. 형태소란 ' 의미를 가진 가장 작은 단위'를 뜻합니다. 그리고 형태학(morphology)이란, 형태소로부터 단어들을 만들어가는 학문을 뜻합니다.
- 형태소는 다음과 같이 두 가지 종류가 있다.
  - 1) 어간(stem): 단어의 의미를 담고 있는 단어의 핵심 부분.
  - 2) 접사(affix): 단어에 추가적인 의미를 주는 부분.

- 형태학적 파싱은 이 두 가지 구성 요소를 분리하는 작업을 말한다. 가령, cats라는 단어에 대해 형태학적 파싱을 수행한다면, 형태학적 파싱은 결과로 cat(어간)와 -s(접사)를 분리한다. 꼭 두 가지로 분리되지 않는 경우도 간혹 있다. 단어 fox는 형태학적 파싱을 한다고 하더라도 더 이상 분리할 수 없는데, fox는 독립적인 형태소이기 때문이고, 이와 유사하게 cat 또한 더 이상 분리되지 않는다.

Text 전처리: 어간 추출(Stemming)

- 어간(Stem)을 추출하는 작업을 어간 추출(stemming)이라고 한다. 어간 추출은 형태학적 분석을 단순화한 버전이라고 볼 수도 있고, 정해진 규칙만 보고 단어의 어미를 자르는 어림짐작의 작업이라고 볼 수도 있다. 다시 말해, 이 작업은 섬세한 작업이 아니기 때문에 어간 추출 후에 나오는 결과 단어는 사전에 존재하지 않는 단어일 수도 있다.

Text 전처리: 불용어(Stopping words)

- 갖고 있는 데이터에서 유의미한 단어 토큰만을 선별하기 위해서는 큰 의미가 없는 단어 토큰을 제거하는 작업이 필요하다. 여기서 큰 의미가 없다라는 것은 자주 등장하지만 분석을 하는 것에 있어서는 큰 도움이 되지 않는 단어들을 말한다.
- 예를 들면, I, my, me, over, 조사, 접미사 같은 단어들은 문장에서는 자주 등장하지만 실제 의미 분석을 하는데는 거의 기여하는 바가 없는 경우가 있다. 이러한 단어들을 불용어(stopword)라고 하며, NLTK에서는 위와 같은 100여개 이상의 영어 단어들을 불용어로 패키지 내에서 미리 정의하고 있다.
- 물론 불용어는 개발자가 직접 정의할 수도 있다.

Text 전처리 실습: nltk

- 갖고 있는 데이터에서 유의미한 단어 토큰만을 선별하기 위해서는 큰 의미가 없는 단어 토큰을 제거하는 작업이 필요하다. 여기서 큰 의미가 없다라는 것은 자주 등장하지만 분석을 하는 것에 있어서는 큰 도움이 되지 않는 단어들을 말한다.
- 예를 들면, I, my, me, over, 조사, 접미사 같은 단어들은 문장에서는 자주 등장하지만 실제 의미 분석을 하는데는 거의 기여하는 바가 없는 경우가 있다. 이러한 단어들을 불용어(stopword)라고 하며, NLTK에서는 위와 같은 100여개 이상의 영어 단어들을 불용어로 패키지 내에서 미리 정의하고 있다.
- 물론 불용어는 개발자가 직접 정의할 수도 있다.

Text 전처리 실습: nltk

- 엔엘티케이(NLTK)는 자연어 처리를 위한 파이썬 패키지이다. 아나콘다를 설치하였다면 NLTK는 기본적으로 설치가 되어져 있습니다. 아나콘다를 설치하지 않았다면 아래의 커맨드로 NLTK를 별도 설치할 수 있다.

- 매우 많은 기능을 제공하므로 타 통계부분에 관심이 있으시면 documentation 참고 추천
  - NLTK :: Natural Language Toolkit

- 설치는 'pip install nltk' 으로 설치.
- Nltk 패키지에는 각각 클래스가 많기때문에 아래와 같이 하나하나 불러온다.

```
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import WordPunctTokenizer
from nltk import pos_tag
```

임베딩이란?

- 아래 표와 같이 문자(str)는 컴퓨터가 해석할 때 그냥 기호일 뿐이다. 이렇게 encoding된 상태로 보게 되면 아래와 같은 문제점이 발생할 수 있다.

컴퓨터가 보는 문자

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

컴퓨터는 ASCII, 유니코드, UTF-8 encoding 등으로 문자를 표현하고 저장한다.

컴퓨터가 보는 단어

단어	1	2	3	4
love	0x6C	0x6F	0x76	0x65
live	0x6C	0x69	0x76	0x65
like	0x6C	0x69	0x6B	0x65

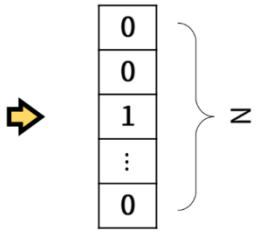
love는 live보다 like와 더 유사하다. 하지만 컴퓨터가 보기에는 love는 live와 더 비슷해 보인다.

임베딩이란?

- 이 글자가 어떤 글자인지를 표시할 수 있고 그에 따른 특성을 갖게 하려면 우선 계산할 수 있게 숫자로 만들어 주어야 할 것이다. 그러한 방법 중 가장 단순한 방법이 One-hot encoding을 통한 것이다. 허나, 이러한 Sparse matrix를 통한 계산은 너무 비효율 적이다. 그렇다면 어떻게 dense하게 표현할 수 있을지를 고민하는 것이 바로 Embedding이라는 개념의 본질일 것이다.

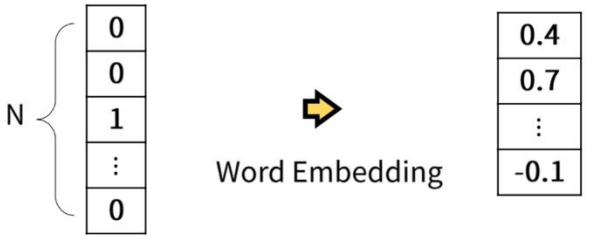
One-Hot Encoding

Code	Word
0	NOT USED
1	i
2	you
3	study
4	fast
5	campus
...	...
N-1	zzz



N개의 단어를 좌측의 코드로 표현하면 희소 표현(Sparse representation)이라고 하며, 우측의 벡터로 표현할 경우 One-Hot Encoding이라고 한다.

밀집 표현 Dense Representation



희소 표현된 단어를 임의의 길이의 실수 벡터로 표현할 경우, 이를 밀집 표현(Dense Representation)이라고 한다. 이 과정을 Word Embedding이라고 하며, 밀집 표현된 결과를 임베딩 벡터(Embedding Vector)라고 부른다.

임베딩이란?

- 자연어 처리(Natural Language Processing)분야에서 임베딩(Embedding)은 **사람이 쓰는 자연어를 기계가 이해할 수 있는 숫자 형태인 vector로 바꾼 결과 혹은 그 일련의 과정 전체를 의미**한다.
- 가장 간단한 형태의 임베딩은 단어의 빈도를 그대로 벡터로 사용하는 것인데, 다음의 예시를 보자. 단어-문서 행렬(Term-Document Matrix)는 row는 단어 column은 문서에 대응한다.

구분	메밀꽃 필 무렵	운수 좋은 날	사랑 손님과 어머니	삼포 가는 길
기차	0	2	10	7
막걸리	0	1	0	0
선술집	0	1	0	0

막걸리-선술집 간 의미의 차이가 막걸리-기차보다 작을 것이라고 추정해 볼 수 있다.

- 위의 표에서 “운수 좋은 날”이라는 문서의 임베딩은 [2, 1, 1]이다. 막걸리라는 단어의 임베딩은 [0, 1, 0, 0]이다. 또한 사랑 손님과 어머니, 삼포 가는 길이 사용하는 단어 목록이 상대적으로 많이 겹치고 있는 것을 알 수 있다. 위의 Matrix를 바탕으로 “사랑 손님과 어머니”는 “삼포 가는 길”과 기차라는 소재를 공유한다는 점에서 비슷한 작품일 것이라는 추정을 해볼 수 있다. 또 막걸리라는 단어와 선술집이라는 단어가 운수 좋은 날이라는 작품에만 등장하는 것을 알 수 있다.

임베딩의 역할

- 단어/문장 간 관련도 계산
- 의미적/문법적 정보 함축
- 전이학습(transfer learning)

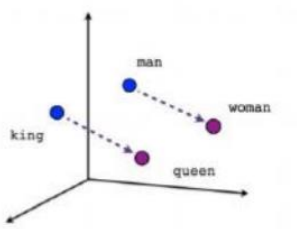
임베딩의 역할

- 단어/문장 간 관련도 계산
  - 단어-문서 행렬은 가장 단순한 형태의 임베딩이다. 현업에서는 이보다 복잡한 형태의 임베딩을 사용한다. 대표적인 임베딩 기법은 word2vec 있고, 컴퓨터가 계산하기 쉽도록 단어를 전체 단어들 간의 관계에 맞춰 해당 단어의 특성을 갖는 벡터로 바꾸면 단어들 사이의 유사도를 계산하는 일이 가능해진다.
  - 자연어일 때 불가능했던 유사도를 계산할 코사인 유사도 계산이 임베딩 덕분에 가능하다는 것이다. 또한 임베딩을 수행하면 벡터 공간을 기하학적으로 나타낸 시각화 역시 가능하다.

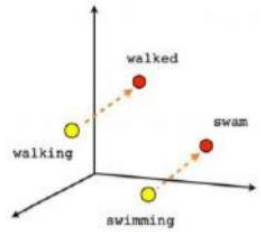
임베딩의 역할

- 의미적/문법적 정보 함축
  - 임베딩은 벡터인 만큼 사칙 연산이 가능하다. 단어 벡터 간 덧셈/뺄셈을 통해 단어들 사이의 의미적, 문법적 관계를 도출해낼 수 있다.
  - 예를 들면, 아들-딸+소녀=소년이 성립하면 성공적인 임베딩이라고 볼 수 있다.
  - 아들 - 딸 사이의 관계와 소년-소녀 사이의 의미 차이가 임베딩에 함축되어 있으면 품질이 좋은 임베딩이다. 이런 임베딩 평가는 단어 유추 평가라고 부른다.

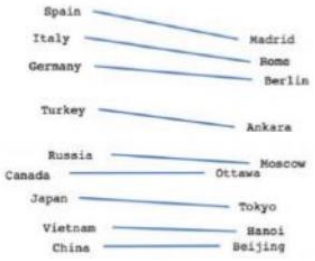
- Woman + (king – man) = queen
- Walked – walking + swimming = swam
- Germany – Berlin + Rome = Italy



Male-Female



Verb tense



Country-Capital

임베딩의 역할

- 전이학습(transfer learning)
  - 품질 좋은 임베딩은 모형의 성능과 모형의 수렴속도가 빨라지는데 이런 품질 좋은 임베딩을 다른 딥러닝 모델의 입력 값으로 사용하는 것을 NLP에서 전이학습이라고 한다.
  - 예를 들어, 대규모 말뭉치를 활용하여 임베딩을 미리 만들고, 임베딩에는 의미적/문법적 정보 등을 녹여 낼수 있다. 이 임베딩을 입력 값으로 쓰는 전이 학습 모델은 문서 분류라는 업무를 빠르게 잘 할 수 있는 것이다.
  - 대표적으로 pretrained BERT가 있다.

문서 임베딩 기법: Term Frequency Vector

- Bag of Words(BoWs)는 TF 벡터를 만드는 가장 기본적인 모델이다. 텍스트를 숫자 형식으로 변환 하는 것.
- 이 모델은 문서 내 모든 단어에서 어휘를 추출하고 문서-용어 행렬을 사용해 모델을 구축한다.
- 장점은 단순하고 이해하기 쉽지만, 단점으로는 모든 단어가 포함된 사전 제작 시간이 오래 걸리고 차원 수가 크다.

Term Document	python	data	hog dogs	baseball	zoo	zebra
#1	1	3	0	0	0	0
#2	0	0	3	2	0	0
#3	3	0	0	0	2	2