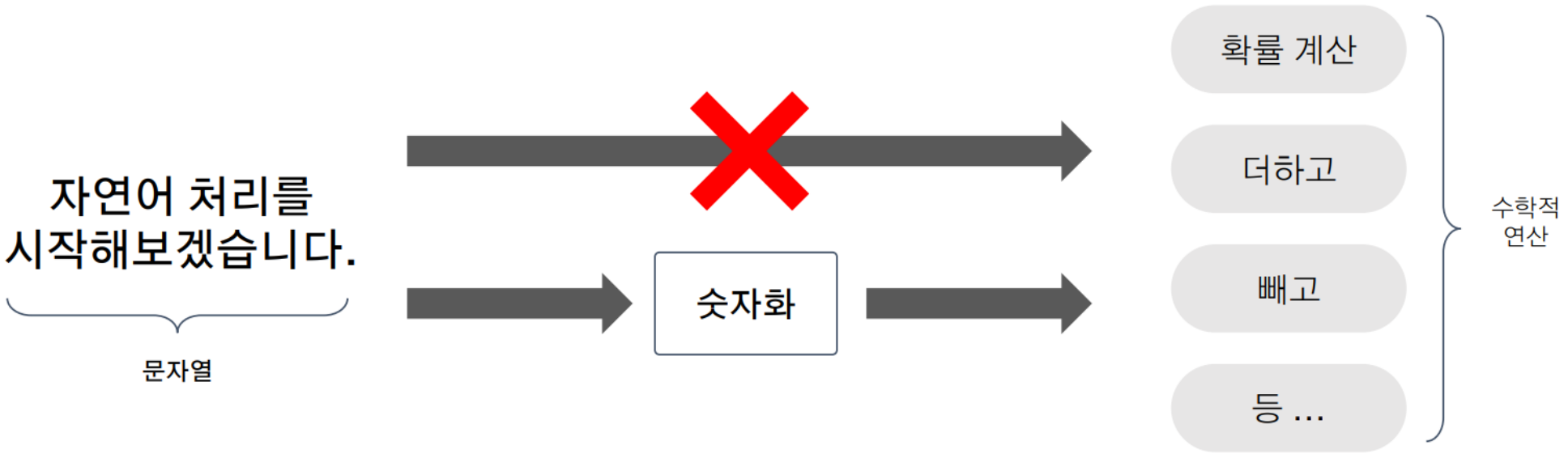
NLP – 단어의 표현

Index

Word Representation

단어의 표현

단어의 표현이 필요한 이유?



One-Hot-Encoding

원핫-인코딩

원핫-인코딩

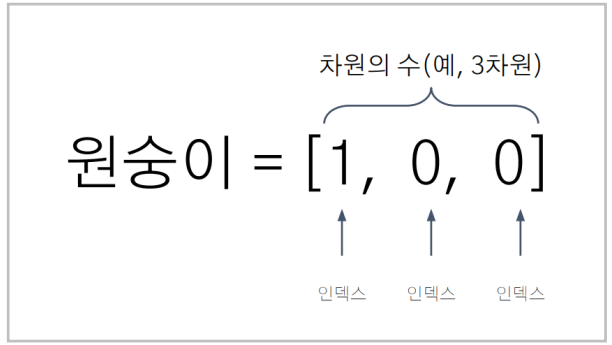
- 원핫-인코딩은 단어(word)를 숫자료 표현하고자 할 때 적용할 수 있는 간단한 방법론

원숭이, 바나나, 사과를 표현할 때

원숭이 = [1,0,0]
바나나 = [0,1,0]
사과 = [0,0,1]

원숭이, 바나나, 사과, 코끼리를 표현할 때

원숭이 = [1,0,0,0]
바나나 = [0,1,0,0]
사과 = [0,0,1,0]
코끼리 = [0,0,0,1]



원핫-인코딩의 한계점

- 차원 크기의 문제

원숭이, 바나나, 사과를 표현할 때

원숭이 = [1,0,0]
바나나 = [0,1,0]
사과 = [0,0,1]

단어의 수만큼 차원이 필요함

단어수가 많아진다면?

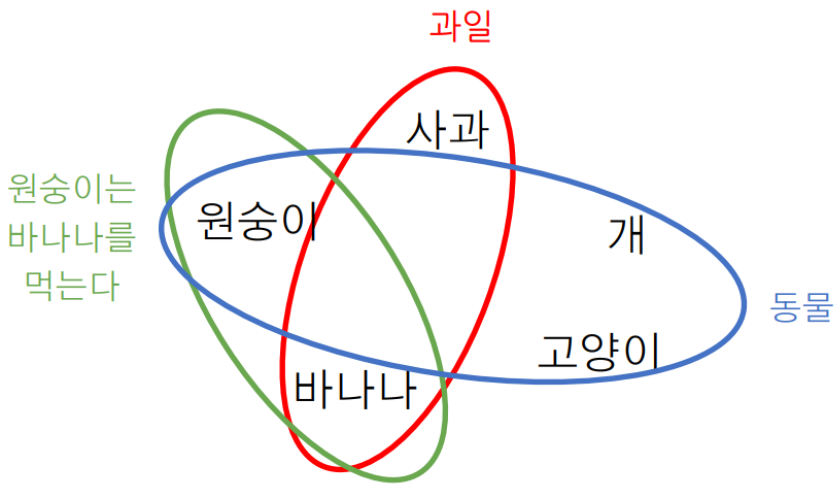
2017년 표준국어대사전에 등재된 단어 수 약 50만개
➔ 50만개의 차원이 필요

원숭이 = [1,0,0,0,0,0,0,0,0, ...,0,0,0,0,0]

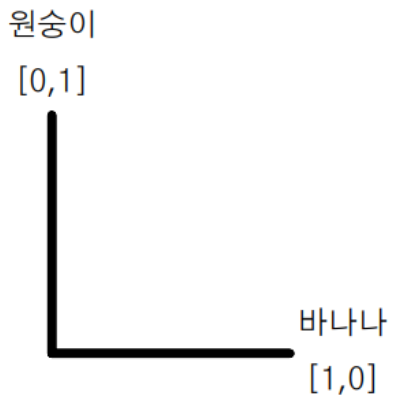
50만 차원 벡터

원핫-인코딩의 한계점

- 의미를 담지 못하는 문제



$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

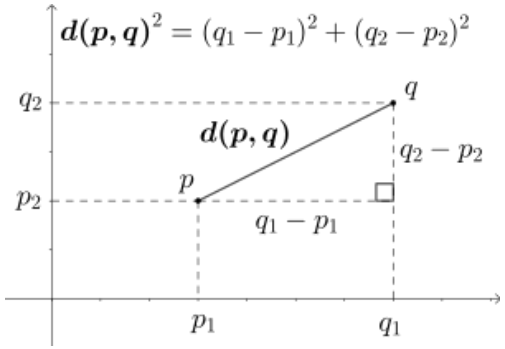


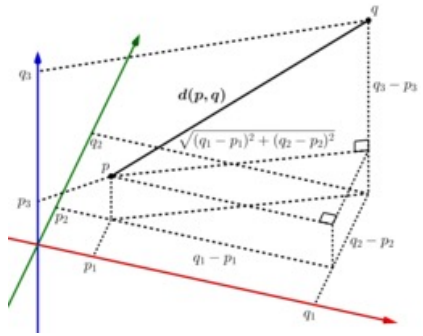
$$\text{similarity} = \frac{(0 \times 1) + (1 \times 0)}{(1^2 + 0^2) \times (0^2 + 1^2)} = 0$$

- 원핫 벡터간의 코사인 유사도는 모두 0
- 따라서, 단어 간 의미를 분간하기 어렵다.

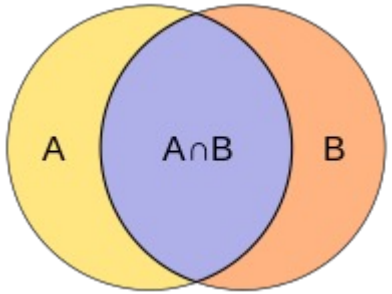
유사도 계산 (Text Similarity)

- 유클리디언 거리 (Euclidean distance)
  → 벡터 간 직진거리를 계산

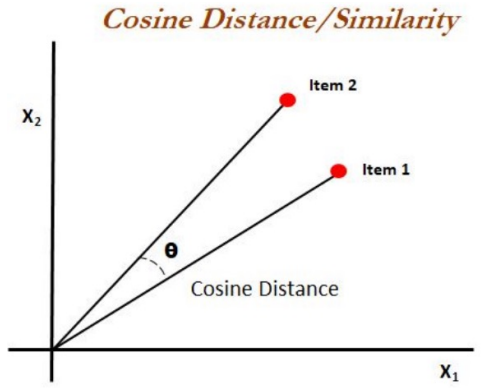




- 자카드 유사도 (Jaccard index)
  → 문서 혹은 문장 간 유사도 측정 (겹치는 토큰의 비율)



- 코사인 유사도 (Cosine Similarity)
  → 두 벡터간 각을 이용한 유사도 측정



- 0도 = 1, 90도 = 0, 180도 = -1
  → 1에 가까울수록 유사도 높음 (= 유사한 의미)

TF-IDF, N-gram

TF-IDF, N-gram

TF-IDF (Term Frequency-Inverse Document Frequency)

- 단어 빈도 - 역문서 빈도
- TDM 내 각 단어의 중요성을 가중치로 표현
- TDM을 사용하는 것보다 더 정확하게 문서비교가 가능

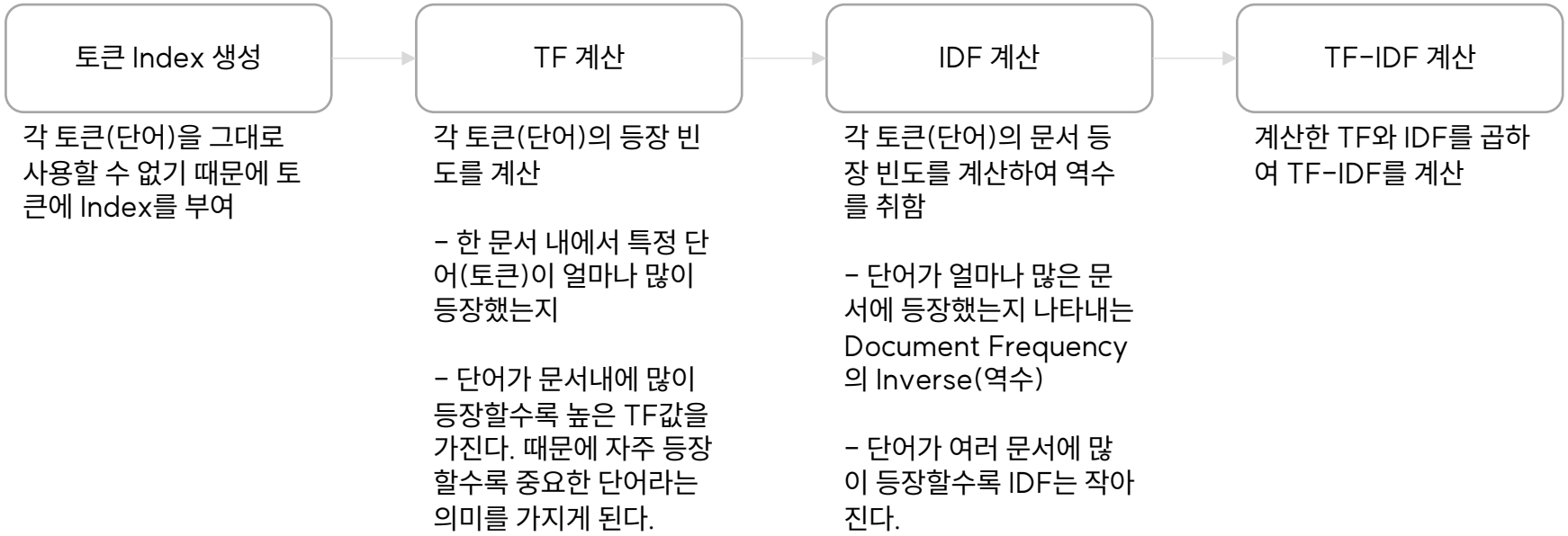
*TDM - 단어 문서 행렬 (Term Document Matrix)

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

tf(d,t)	특정 문서 d에서 특정 단어 t의 등장 횟수
df(t)	특정 단어 t가 등장한 문서의 수
idf(d, t)	df(t)의 역수

TF	IDF	TF-IDF	설명
높	높	높	특정 문서에 많이 등장하고 타 문서에 많이 등장하지 않는 단어 (중요 키워드)
높	낮	-	특정 문서에도 많이 등장하고 타 문서에도 많이 등장하는 단어
낮	높	-	특정 문서에는 많이 등장하지 않고 타 문서에만 많이 등장하는 단어
낮	낮	낮	특성 문서에 많이 등장하지 않고 타 문서에만 많이 등장하는 단어

TF-IDF 계산 과정


토큰 Index 생성
각 토큰(단어)을 그대로 사용할 수 없기 때문에 토큰에 Index를 부여
TF 계산
각 토큰(단어)의 등장 빈도를 계산
- 한 문서 내에서 특정 단어(토큰)이 얼마나 많이 등장했는지
- 단어가 문서내에 많이 등장할수록 높은 TF값을 가진다. 때문에 자주 등장할수록 중요한 단어라는 의미를 가지게 된다.
IDF 계산
각 토큰(단어)의 문서 등장 빈도를 계산하여 역수를 취함
- 단어가 얼마나 많은 문서에 등장했는지 나타내는 Document Frequency의 Inverse(역수)
- 단어가 여러 문서에 많이 등장할수록 IDF는 작아진다.
TF-IDF 계산
계산한 TF와 IDF를 곱하여 TF-IDF를 계산

예제 1

문서 1: d1 = “The cat sat on my face I hate a cat”
문서 2: d2 = “The dog sat on my bed I love a dog”

1. 토큰 Index 생성

token	index
The	0
cat	1
sat	2
on	3
my	4
face	5
I	6
hate	7
a	8
dog	9
bed	10
lov	11

2. TF 계산

$$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$$

$f_{t,d}$ = 문서내 토큰 빈도

SUM($f_{t,d}$)= 문서내 전체 토큰빈도

문서 1

token	문서 내 토큰 빈도	전체 토큰 수	TF
The	1	10	0.1
cat	2	10	0.2
sat	1	10	0.1
on	1	10	0.1
my	1	10	0.1
face	1	10	0.1
I	1	10	0.1
hate	1	10	0.1
a	1	10	0.1
dog	0	10	0
bed	0	10	0
lov	0	10	0

문서 2

token	문서 내 토큰 빈도	전체 토큰 수	TF
The	1	10	0.1
cat	0	10	0
sat	1	10	0.1
on	1	10	0.1
my	1	10	0.1
face	0	10	0
I	1	10	0.1
hate	0	10	0
a	1	10	0.1
dog	2	10	0.2
bed	1	10	0.1
lov	1	10	0.1

- TF 가중치 계산하는 여러가지 방법

Variants of term frequency (tf) weight

weighting scheme	tf weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

2. IDF 계산

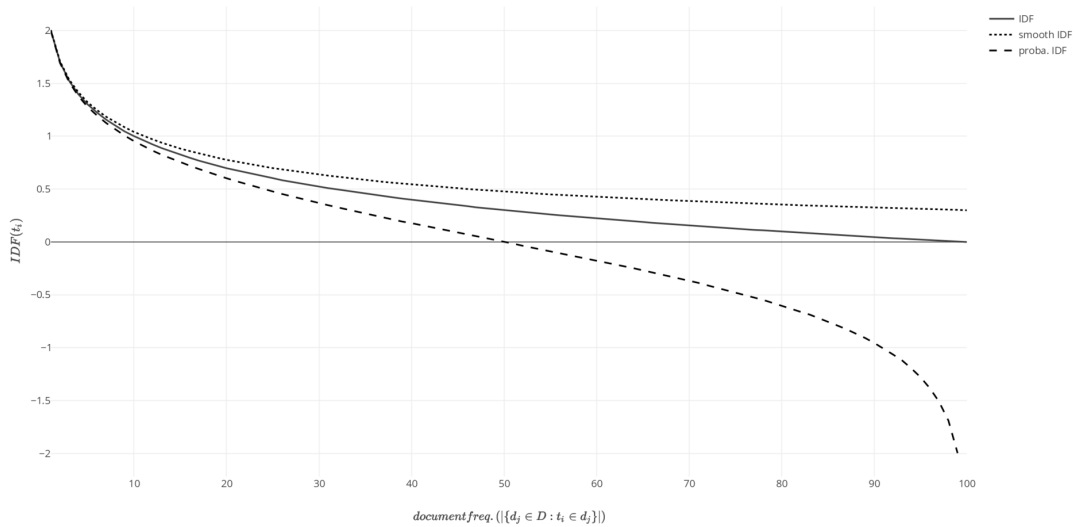
$$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$$

N = 문서수

n_t = 토큰이 등장한 문서수

token	문서 수	토큰 등장한 문서 수	IDF
The	2	2	0
cat	2	1	0.301
sat	2	2	0
on	2	2	0
my	2	2	0
face	2	1	0.301
I	2	2	0
hate	2	1	0.301
a	2	2	0
dog	2	1	0.301
bed	2	1	0.301
lov	2	1	0.301

IDF에 로그를 사용하는 이유



- 총 문서의 수가 증가할수록 IDF의 값이 기하급수적으로 커짐

- IDF 가중치 계산하는 여러가지 방법

Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ($n_t = \left\vert\{d \in D : t \in d\}\right\vert$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(\frac{N}{1+n_t}\right)$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1+n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N-n_t}{n_t}$

3. TF-IDF 계산

문서 1

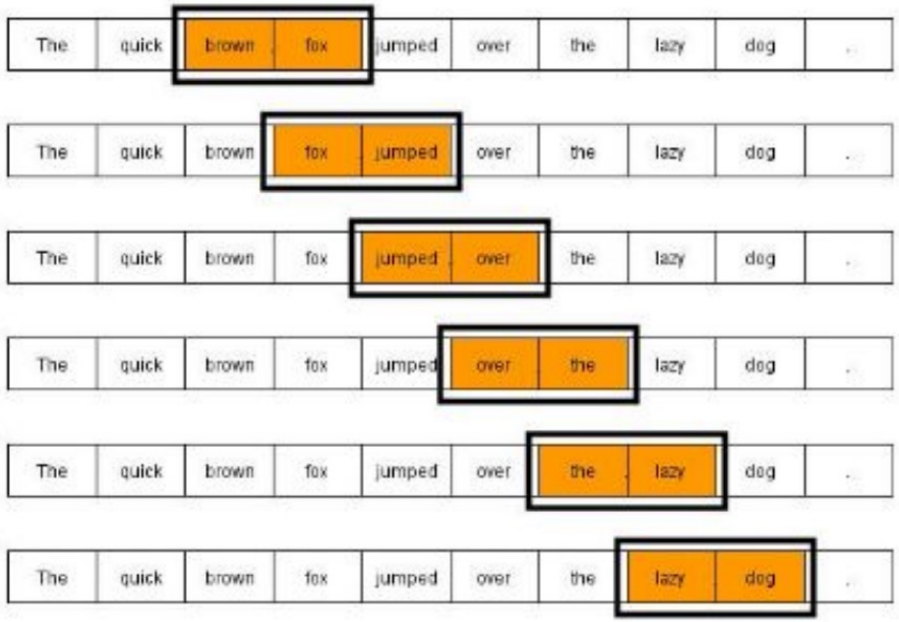
token	TF	IDF	TF-IDF
The	0.1	0	0
cat	0.2	0.301	0.060
sat	0.1	0	0
on	0.1	0	0
my	0.1	0	0
face	0.1	0.301	0.030
I	0.1	0	0
hate	0.1	0.301	0.030
a	0.1	0	0
dog	0	0.301	0
bed	0	0.301	0
lov	0	0.301	0

문서 2

token	TF	IDF	TF-IDF
The	0.1	0	0
cat	0	0.301	0
sat	0.1	0	0
on	0.1	0	0
my	0.1	0	0
face	0	0.301	0
I	0.1	0	0
hate	0	0.301	0
a	0.1	0	0
dog	0.2	0.301	0.060
bed	0.1	0.301	0.030
lov	0.1	0.301	0.030

N-gram

- 몇 개(n) 단어를 함께 보느냐에 따라 unigram, bigram, trigram등으로 구분
- 제한적으로 문맥을 표현할 수 있음



N-gram

an adorable little boy is spreading smile

- unigram : an, adorable, little, boy, is, spreading, smiles
- bigrams : an adorable, adorable little, little boy, boy is, is spreading, spreading smiles
- trigrams : an adorable little, adorable little boy, little boy is, boy is spreading, is spreading smiles
- 4-grams : an adorable little boy, adorable little boy is, little boy is spreading, boy is spreading smiles

N-gram의 한계

- n의 크기는 trade-off 문제
  - 1보다는 2를 선택하는 것이 모델 성능을 높일 수 있음
  - n을 너무 크게 선택하면 n-gram 이 unique 할 확률이 높아 등장수가 낮을 확률이 높음 (Out of Vocabulary 문제)
  - n을 너무 작게 하면 카운트는 잘되지만 정확도가 떨어질 수 있음. n은 최대 5를 넘지 않도록 권장

- N-gram 카운트가 0인 경우
  - N-gram이 모든 단어를 커버할 수 없기 때문에 OOV 문제가 발생할 수 있음

적용 분야에 맞는 corpus 수집

- 분야에 따라 단어들의 확률 분포는 다름 (금융 분야, 마케팅 분야)
- 분야에 적합한 코퍼스를 사용하면 언어 모델의 성능이 높아질 수 있음

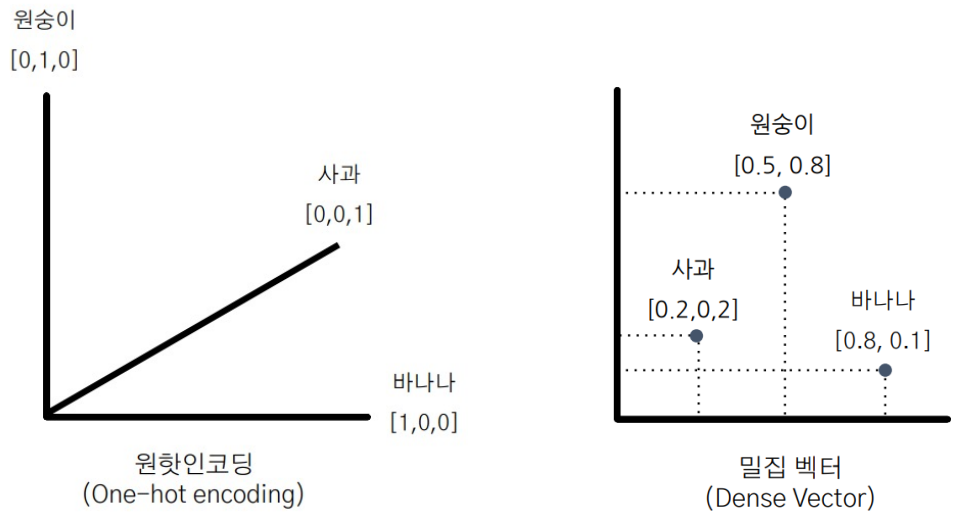
Word Embedding

단어 임베딩

단어 임베딩 (Word Embedding)

– 단어 임베딩은 단어의 의미를 간직하는 밀집 벡터(Dense Vector)로 표현하는 방법

원숭이, 바나나, 사과를 표현할 때



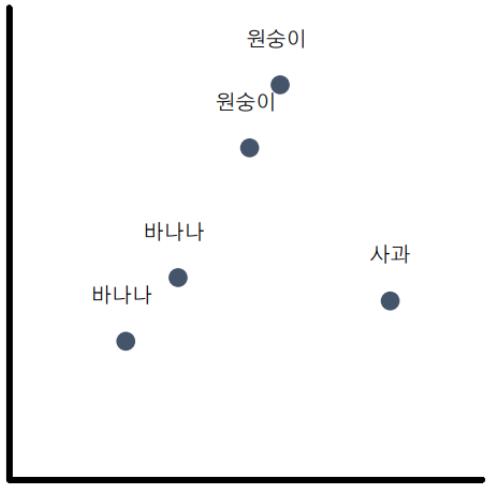
– 새로운 단어 추가 시 차원을 추가할 필요 없음
➔ 차원을 줄일 수 있음
➔ 추후 분류나 예측 모델을 학습할 때 연산을 줄일 수 있는 이점

단어 임베딩의 한계

- 차원의 크기는 밀집벡터로 해결했음
- 하지만 단어 의미는?

분포 가설

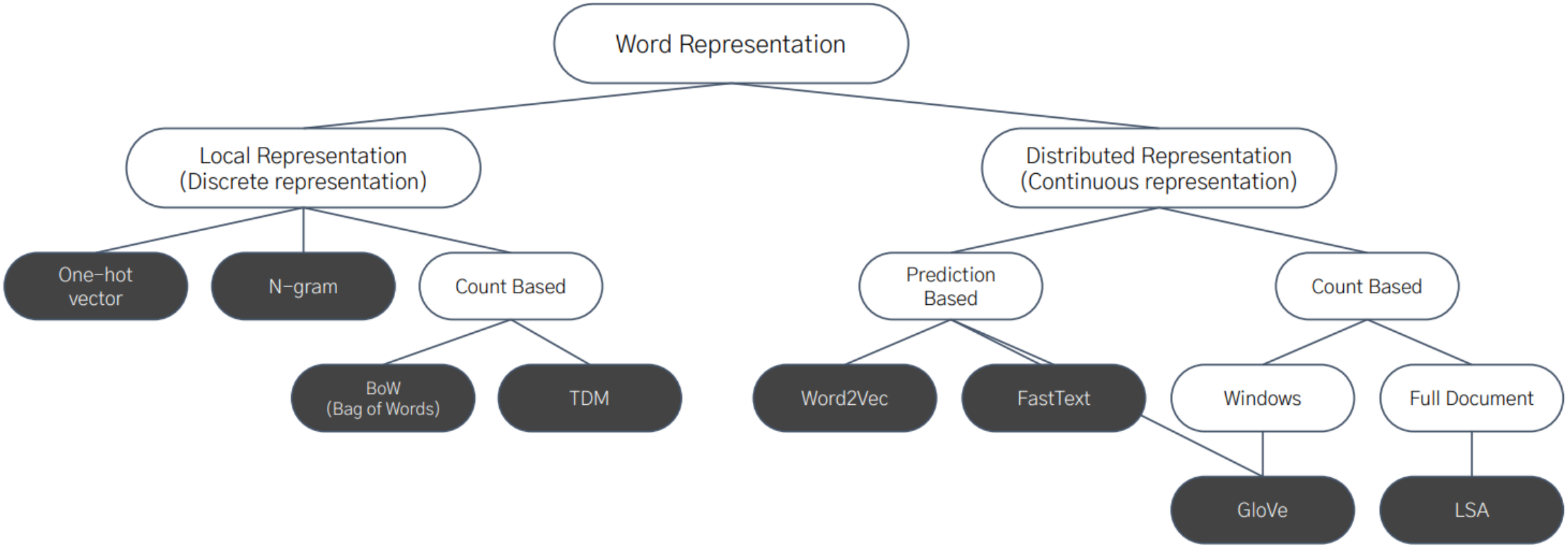
- 같은 문맥에서 등장하는 단어는 유사한 의미를 지닌다.



1) 임의의 위치에 벡터 생성

2) 같은 문맥이 등장하는 단어를 더 가까이 표현

Word Representation



- Local representation (Discrete representation): 해당 단어 그 자체만 보고 값을 매핑하여 표현
- Distributed representation (Continuous representation): 단어를 표현하기 위해 주변을 참고