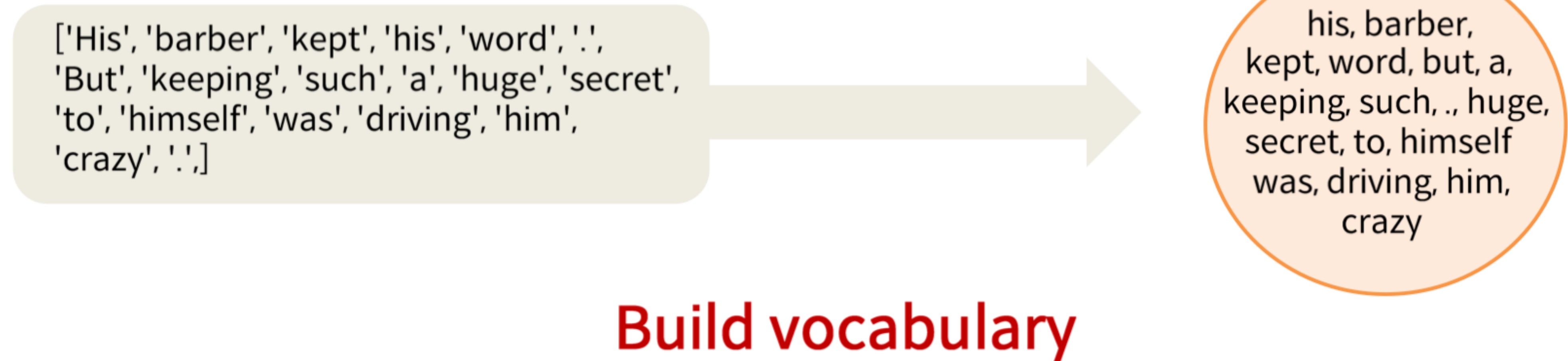


# 텍스트의 벡터화

# Vocabulary 의 생성

기계가 알고 있는 단어들의 집합을 단어 집합(Vocabulary)이라고 한다.

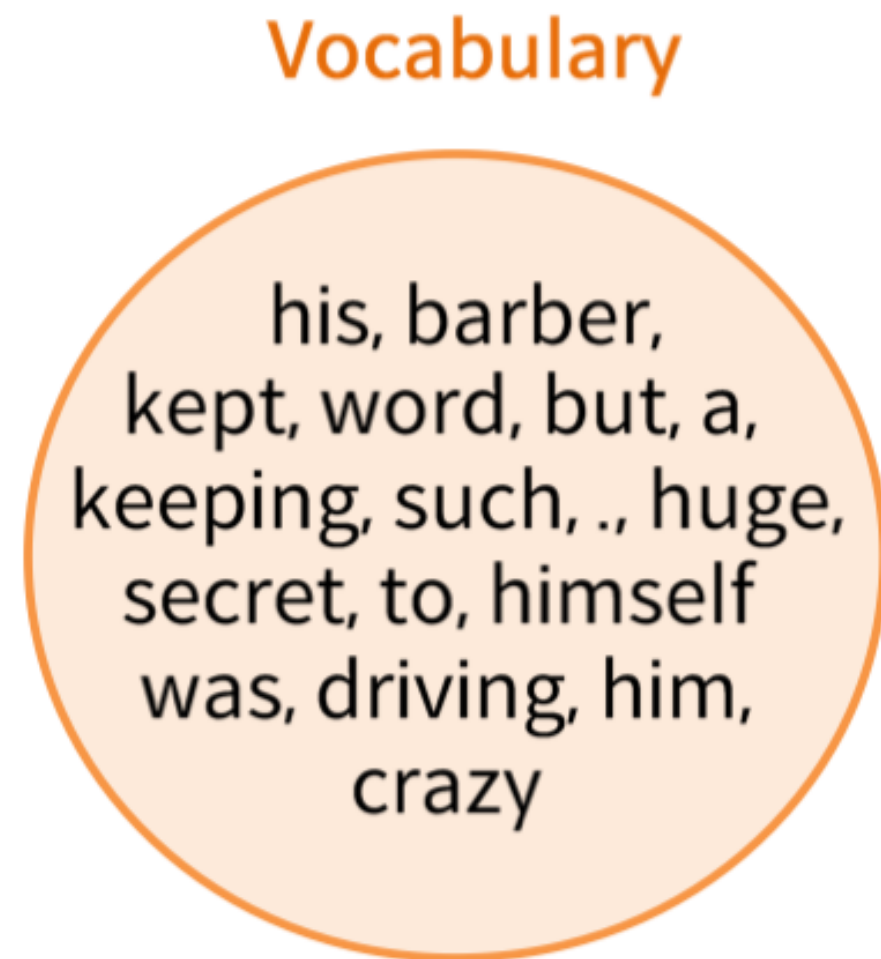
단어 집합이란 훈련 데이터에 있는 **단어들의 중복을 제거한 집합**을 의미한다.



# Vocabulary 의 생성

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.



{'his': 1, 'barber': 2, 'kept': 3,  
'word': 4, 'but': 5, 'a': 6, 'keeping': 7,  
'such': 8, '.': 9, 'huge': 10, 'secret': 11,  
'to': 12, 'himself': 13, 'was': 14,  
'driving': 15, 'him': 16, 'crazy': 17}

각 단어에 정수가 부여된다.  
단어 집합을 기반으로 하므로 중복은 허용되지 않는다.

# Vocabulary 의 생성

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

## Integer Encoding

새로운 문장이 입력!

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his': 1, 'barber': 2, 'kept': 3,  
'word': 4, 'but': 5, 'a': 6, 'keeping': 7,  
'such': 8, '.': 9, 'huge': 10, 'secret': 11,  
'to': 12, 'himself': 13, 'was': 14,  
'driving': 15, 'him': 16, 'crazy': 17}



# Vocabulary 의 생성

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

## Integer Encoding

새로운 문장이 입력!

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his': 1, 'barber': 2, 'kept': 3,  
'word': 4, 'but': 5, 'a': 6, 'keeping': 7,  
'such': 8, '.': 9, 'huge': 10, 'secret': 11,  
'to': 12, 'himself': 13, 'was': 14,  
'driving': 15, 'him': 16, 'crazy': 17}

['1', '2', '3', '6', '11', '9']

각 단어를 고유한 정수로!

# Vocabulary 의 생성

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

## Integer Encoding

여러 문장에 대해서는?

[[ 'his', 'barber', 'kept', 'a', 'secret', '.' ],  
[ 'a', 'barber', 'was', 'driving', '.' ]]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,  
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,  
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,  
'to' : 12, 'himself' : 13, 'was' : 14,  
'driving' : 15, 'him' : 16, 'crazy' : 17 }

[[ '1', '2', '3', '6', '11', '9'],  
[ '6', '2', '14', '15' ]]

각 단어를 고유한 정수로!

<https://core.today>

# OOV 문제(Out-Of-Vocabulary Problem)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.

이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다

## Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[[ 'his', 'teacher', 'kept', 'a', 'secret', '.' ],  
[ 'a', 'barber', 'was', 'driving', '.' ]]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,  
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,  
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,  
'to' : 12, 'himself' : 13, 'was' : 14,  
'driving' : 15, 'him' : 16, 'crazy' : 17 }

# OOV 문제(Out-Of-Vocabulary Problem)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.

이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다

## Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[[ 'his', 'teacher', 'kept', 'a', 'secret', '.' ],  
[ 'a', 'barber', 'was', 'driving', '.' ]]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,  
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,  
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,  
'to' : 12, 'himself' : 13, 'was' : 14,  
'driving' : 15, 'him' : 16, 'crazy' : 17 }

앞으로 모르는 단어가 오면 특별한 토큰 'UNK'로 맵핑하도록 약속!



# OOV 문제(Out-Of-Vocabulary Problem)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.

이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다

## Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[[ 'his', 'teacher', 'kept', 'a', 'secret', '.' ],  
[ 'a', 'barber', 'was', 'driving', '.' ]]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,  
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,  
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,  
'to' : 12, 'himself' : 13, 'was' : 14,  
'driving' : 15, 'him' : 16, 'crazy' : 17 }  
'unk', : 18

앞으로 모르는 단어가 오면 특별한 토큰 'UNK'로 맵핑하도록 약속!

# OOV 문제(Out-Of-Vocabulary Problem)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.

이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다

## Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[[`'his'`, `'teacher'`, `'kept'`, `'a'`, `'secret'`, `'.'`],  
[`'a'`, `'barber'`, `'was'`, `'driving'`, `'.'`]]

{`'his'` : 1, `'barber'` : 2, `'kept'` : 3,  
`'word'` : 4, `'but'` : 5, `'a'` : 6, `'keeping'` : 7,  
`'such'` : 8, `'.'` : 9, `'huge'` : 10, `'secret'` : 11,  
`'to'` : 12, `'himself'` : 13, `'was'` : 14,  
`'driving'` : 15, `'him'` : 16, `'crazy'` : 17}  
`'unk'` : 18

[[`'1'`, `'18'`, `'3'`, `'6'`, `'11'`, `'9'`],  
[`'6'`, `'2'`, `'14'`, `'15'`]]

각 단어를 고유한 정수로!

<https://core.today>

# OOV 문제(Out-Of-Vocabulary Problem)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.

이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다

## Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[[**'his'**, **'teacher'**, 'kept', 'a', 'secret', '.'],  
['a', 'barber', 'was', 'driving', '.']]

{**'his'** : 1, **'barber'** : 2, **'kept'** : 3,  
**'word'** : 4, **'but'** : 5, **'a'** : 6, **'keeping'** : 7,  
**'such'** : 8, **'.'** : 9, **'huge'** : 10, **'secret'** : 11,  
**'to'** : 12, **'himself'** : 13, **'was'** : 14,  
**'driving'** : 15, **'him'** : 16, **'crazy'** : 17}  
**'unk'**, : 18

앞으로 문장의 길이를 동일하게 해주기 위해서는  
특별한 토큰 'PAD'를 사용하도록 약속!

[[**'1'**, **'18'**, **'3'**, **'6'**, **'11'**, **'9'**],  
[**'6'**, **'2'**, **'14'**, **'15'**]]



# OOV 문제(Out-Of-Vocabulary Problem)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.

이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다

## Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[[**'his'**, **'teacher'**, 'kept', 'a', 'secret', '.'],  
['a', 'barber', 'was', 'driving', '.']]

{**'his'** : 1, **'barber'** : 2, **'kept'** : 3,  
**'word'** : 4, **'but'** : 5, **'a'** : 6, **'keeping'** : 7,  
**'such'** : 8, **'.'** : 9, **'huge'** : 10, **'secret'** : 11,  
**'to'** : 12, **'himself'** : 13, **'was'** : 14,  
**'driving'** : 15, **'him'** : 16, **'crazy'** : 17}  
**'unk'** : 18, **'pad'** : 0

앞으로 문장의 길이를 동일하게 해주기 위해서는  
특별한 토큰 'PAD'를 사용하도록 약속!

[[**'1'**, **'18'**, **'3'**, **'6'**, **'11'**, **'9'**],  
[**'6'**, **'2'**, **'14'**, **'15'**]]



# OOV 문제(Out-Of-Vocabulary Problem)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.

이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다

## Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[[**'his'**, **'teacher'**, 'kept', 'a', 'secret', '.'],  
['a', 'barber', 'was', 'driving', '.']]

{**'his'** : 1, **'barber'** : 2, **'kept'** : 3,  
**'word'** : 4, **'but'** : 5, **'a'** : 6, **'keeping'** : 7,  
**'such'** : 8, **'.'** : 9, **'huge'** : 10, **'secret'** : 11,  
**'to'** : 12, **'himself'** : 13, **'was'** : 14,  
**'driving'** : 15, **'him'** : 16, **'crazy'** : 17}  
**'unk'**, : 18, **'pad'**: 0

패딩 결과

[[**'1'**, **'18'**, '3', '6', '11', '9'],  
['6', '2', '14', '15', **'0'**, **'0'**]]

[[**'1'**, **'18'**, '3', '6', '11', '9'],  
['6', '2', '14', '15']]

# 벡터화(Vectorization)

---

## 1. 벡터화에 신경망을 사용하지 않을 경우

- 단어에 대한 벡터 표현 방법: 원-핫 인코딩
- 문서에 대한 벡터 표현 방법 : Document Term Matrix, TF-IDF

## 2. 벡터화에 신경망을 사용하는 경우 (2008 ~ 2018)

- 단어에 대한 벡터 표현 방법 : 워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

## 3. 문맥을 고려한 벡터 표현 방법 : ELMO, BERT, GPT, T5 등 ... (2018 - 현재)

- Pre-trained Language Model의 시대.

# 벡터화(Vectorization)

---

## 1. 벡터화에 신경망을 사용하지 않을 경우

- 단어에 대한 벡터 표현 방법: **원-핫 인코딩**
- 문서에 대한 벡터 표현 방법 : Document Term Matrix, TF-IDF

## 2. 벡터화에 신경망을 사용하는 경우 (2008 ~ 2018)

- 단어에 대한 벡터 표현 방법 : **워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)**
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

## 3. 문맥을 고려한 벡터 표현 방법 : ELMO, BERT, GPT, T5 등 ... (2018 - 현재)

- Pre-trained Language Model의 시대.

# 단어의 벡터화 : 원-핫 인코딩

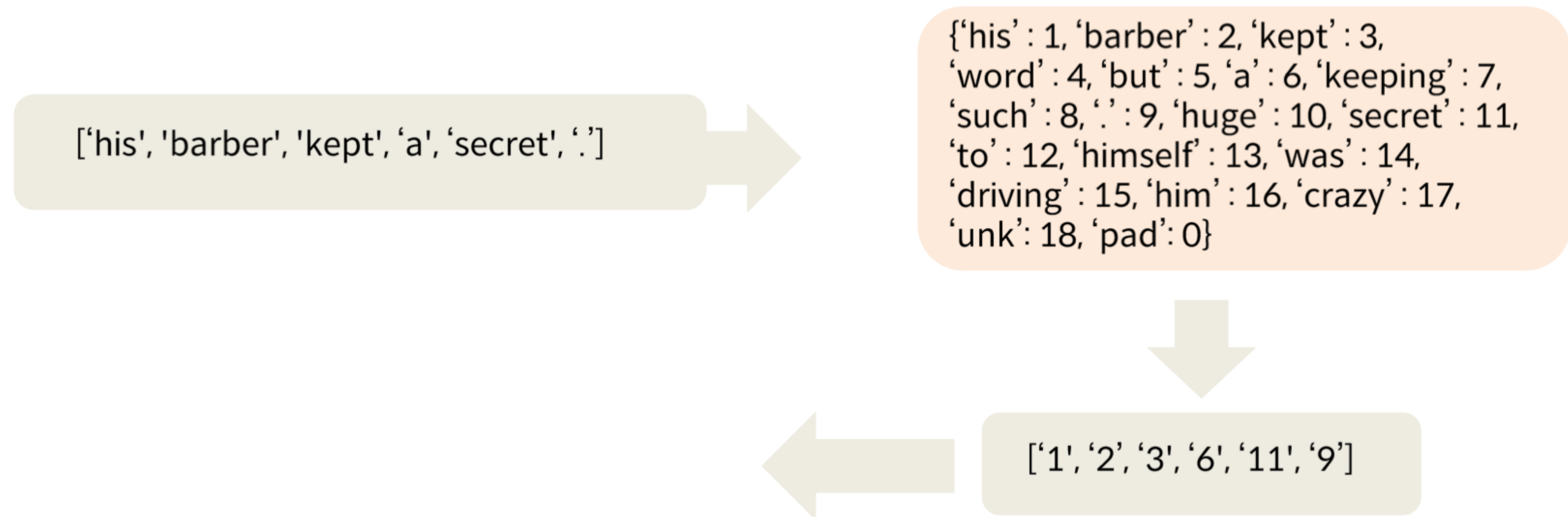
- > 원-핫 인코딩은 전체 단어 집합의 크기를 벡터의 차원으로 가진다.
- > 각 단어에 고유한 정수 인덱스를 부여한 후 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터





## 단어의 벡터화 : 원-핫 인코딩

- > 원-핫 인코딩은 전체 단어 집합의 크기를 벡터의 차원으로 가진다.
- > 각 단어에 고유한 정수 인덱스를 부여한 후 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터.



**원-핫 인코딩!**

# 단어의 벡터화 : 원-핫 인코딩

- > 원-핫 인코딩은 전체 단어 집합의 크기를 벡터의 차원으로 가진다.
- > 각 단어에 고유한 정수 인덱스를 부여한 후 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터.

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his': 1, 'barber': 2, 'kept': 3,  
'word': 4, 'but': 5, 'a': 6, 'keeping': 7,  
'such': 8, '.': 9, 'huge': 10, 'secret': 11,  
'to': 12, 'himself': 13, 'was': 14,  
'driving': 15, 'him': 16, 'crazy': 17,  
'unk': 18, 'pad': 0}

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

['1', '2', '3', '6', '11', '9']

원-핫 인코딩!

# 단어의 벡터화 : 원-핫 인코딩

- > 원-핫 인코딩은 전체 단어 집합의 크기를 벡터의 차원으로 가진다.
- > 각 단어에 고유한 정수 인덱스를 부여한 후 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터.

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his': 1, 'barber': 2, 'kept': 3,  
'word': 4, 'but': 5, 'a': 6, 'keeping': 7,  
'such': 8, '.': 9, 'huge': 10, 'secret': 11,  
'to': 12, 'himself': 13, 'was': 14,  
'driving': 15, 'him': 16, 'crazy': 17,  
'unk': 18, 'pad': 0}

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

벡터의 차원이 단어 집합의 크기!

['1', '2', '3', '6', '11', '9']

원-핫 인코딩!

# 단어의 벡터화 : 원-핫 인코딩

- > 원-핫 인코딩은 전체 단어 집합의 크기를 벡터의 차원으로 가진다.
- > 각 단어에 고유한 정수 인덱스를 부여한 후 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터.

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his': 1, 'barber': 2, 'kept': 3,  
'word': 4, 'but': 5, 'a': 6, 'keeping': 7,  
'such': 8, '.': 9, 'huge': 10, 'secret': 11,  
'to': 12, 'himself': 13, 'was': 14,  
'driving': 15, 'him': 16, 'crazy': 17,  
'unk': 18, 'pad': 0}

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

['1', '2', '3', '6', '11', '9']

단어 벡터 간 유의미한 유사도를 구할 수 없다는 한계가 존재한다... 원-핫 인코딩!



# 벡터화(Vectorization)

---

## 1. 벡터화에 신경망을 사용하지 않을 경우

- 단어에 대한 벡터 표현 방법: **원-핫 인코딩**
- 문서에 대한 벡터 표현 방법 : Document Term Matrix, TF-IDF

## 2. 벡터화에 신경망을 사용하는 경우 (2008 ~ 2018)

- 단어에 대한 벡터 표현 방법 : **워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)**
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

## 3. 문맥을 고려한 벡터 표현 방법 : ELMO, BERT, GPT, T5 등 ... (2018 - 현재)

- Pre-trained Language Model의 시대.

# 단어의 벡터화: 워드 임베딩

- > 인공 신경망을 이용하여 단어의 벡터값을 얻는 방법
- > 학습 후에는 각 단어 벡터 간의 유사도(의미를 반영)를 계산할 수 있다.

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his' : 1, 'barber' : 2, 'kept' : 3,  
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,  
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,  
'to' : 12, 'himself' : 13, 'was' : 14,  
'driving' : 15, 'him' : 16, 'crazy' : 17,  
'unk' : 18, 'pad' : 0}

1.2	0.8	0.1	0.2	0.1	0.5	0.1
0.7	0.2	0.5	2.0	0.7	0.11	0.38
5.8	-0.5	3.7	0.11	-1.5	0.8	0.7
0.2	0.7	1.2	8.1	0.5	0.1	0.2
1.7	2.1	1.1	0.1	7.8	-0.1	0.8
1.2	1.9	2.1	1.1	7.5	-0.9	0.1

['1', '2', '3', '6', '11', '9']

**워드 임베딩!**

# 벡터화(Vectorization)

---

## 1. 벡터화에 신경망을 사용하지 않을 경우

- 단어에 대한 벡터 표현 방법: 원-핫 인코딩
- 문서에 대한 벡터 표현 방법 : **Document Term Matrix, TF-IDF**

## 2. 벡터화에 신경망을 사용하는 경우 (2008 ~ 2018)

- 단어에 대한 벡터 표현 방법 : 워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

## 3. 문맥을 고려한 벡터 표현 방법 : ELMO, BERT, GPT, T5 등 ... (2018 - 현재)

- Pre-trained Language Model의 시대.

# Vectorization : Document Term Matrix

- > DTM은 마찬가지로 벡터가 단어 집합의 크기를 가지며, 대부분의 원소가 0을 가진다.
- > 각 단어는 고유한 정수 인덱스를 가지며, 해당 단어의 등장 횟수를 해당 인덱스의 값으로 가진다.

## Integer encoding

문서1 : 먹고 싶은 사과  
문서2 : 먹고 싶은 바나나  
문서3 : 길고 노란 바나나 바나나  
문서4 : 저는 과일이 좋아요

과일이	1
길고	2
노란	3
먹고	4
바나나	5
사과	6
싶은	7
저는	8
좋아요	9

## Bag of Words 가설 기반 방법론

-	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



# Bag of Words

- > Bag of Words(BoW)를 직역하면 단어들의 가방을 의미한다.
- > 가방에 문장의 단어들을 넣고 흔든다면, 단어의 순서는 무의미해진다.
- > 정리 : 단어의 순서는 무시하고, 오직 단어의 빈도수에만 집중하는 방법

## The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

15



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

# TF-IDF(Term Frequency-Inverse Document Frequency)

- > DTM(Document Term Matrix)에서 추가적으로 중요한 단어에 가중치를 주는 방식
- > TF-IDF 기준으로 중요한 단어는 값이 Up.
- > TF-IDF 기준으로 중요하지 않은 값이 Down.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	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



TF-IDF

-	과일이	길고	노란	먹고	바나나	사과	싫은	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

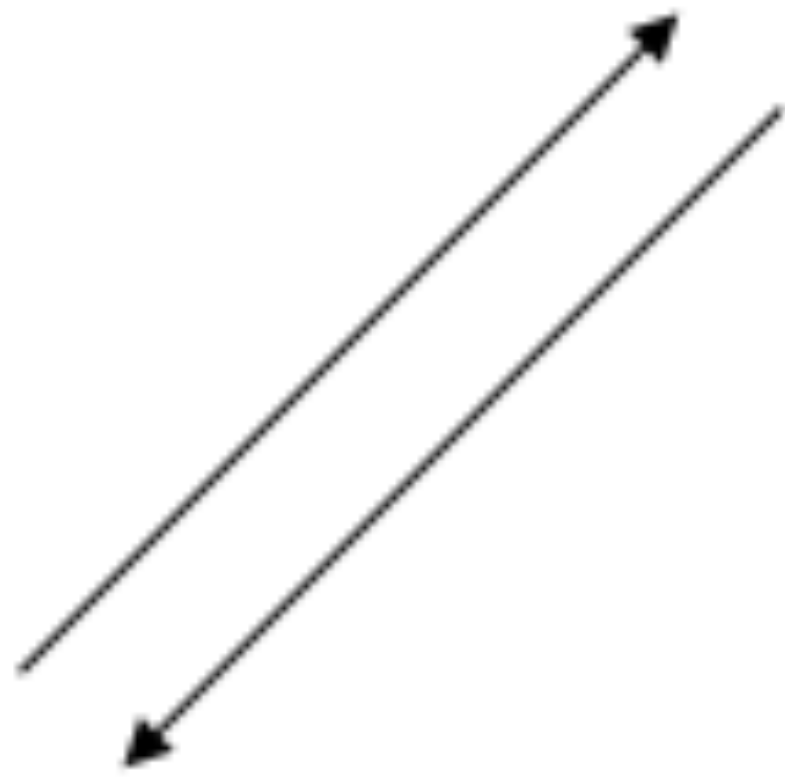
# TF-IDF(Term Frequency-Inverse Document Frequency)

---

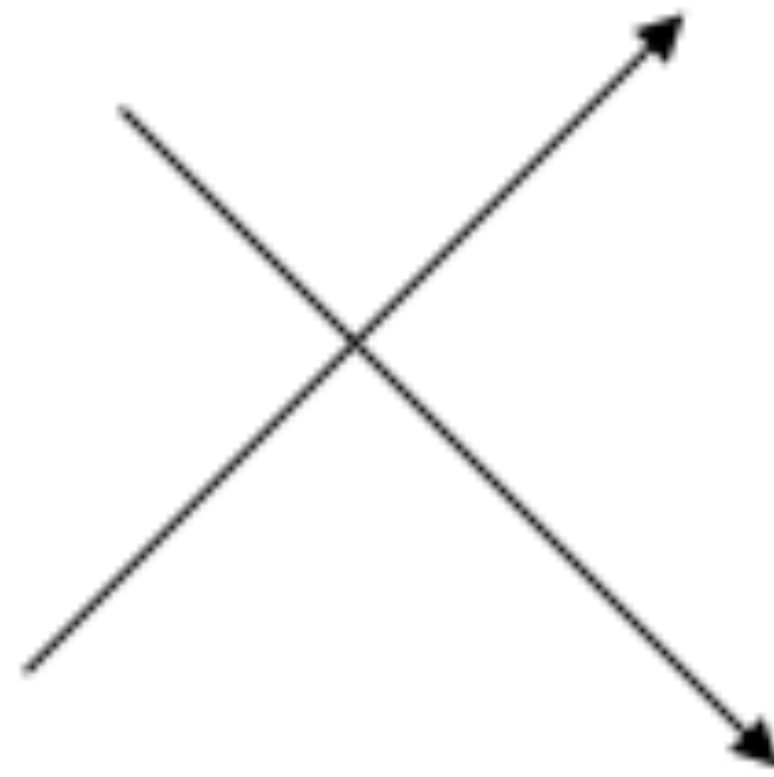
- > TF-IDF는 직역하면 '단어 빈도-역 문서 빈도'
- > TF-IDF는 TF와 IDF라는 두 값을 곱한 결과이다.
- > 문서의 유사도, 검색 시스템에서 검색 결과의 순위 등을 구하는 일에 쓰인다.
- > 벡터이므로 인공 신경망의 입력으로도 사용할 수 있다.

# 코사인 유사도(Cosine Similarity)

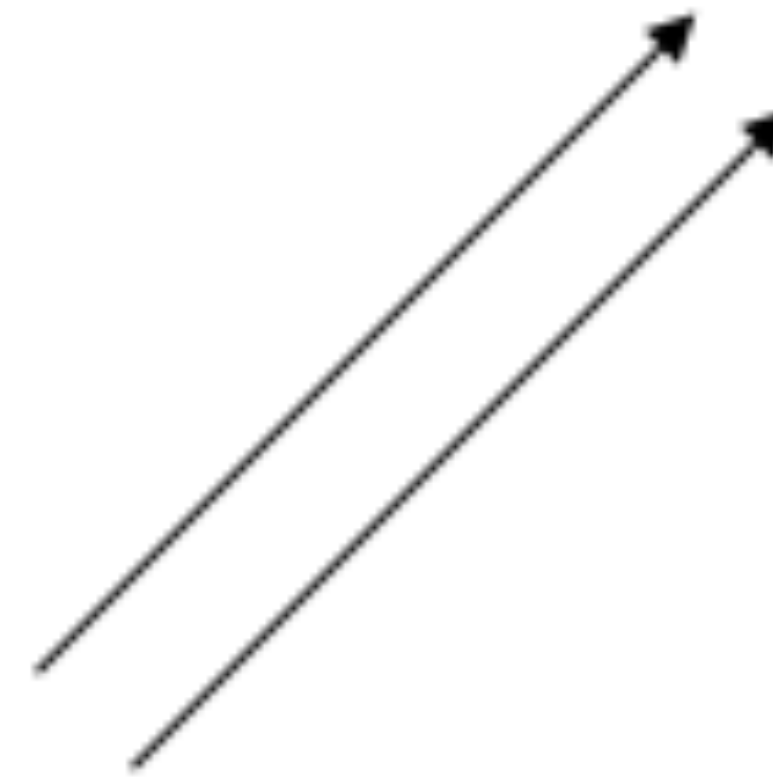
<https://wikidocs.net/24603>



코사인 유사도 : -1



코사인 유사도 : 0



코사인 유사도 : 1

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

# TF-IDF(Term Frequency-Inverse Document Frequency)

---

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- > 다음의 예시로부터 TF-IDF를 계산해보자.

문서1 : 먹고 싶은 사과

문서2 : 먹고 싶은 바나나

문서3 : 길고 노란 바나나 바나나

문서4 : 저는 과일이 좋아요



# TF-IDF(Term Frequency-Inverse Document Frequency)

---

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

>  $tf(d,t)$  : 특정 문서  $d$ 에서의 특정 단어  $t$ 의 등장 횟수.

>  $df(t)$  : 특정 단어  $t$ 가 등장한 문서의 수.

문서1 : 먹고 싶은 사과

문서2 : 먹고 싶은 바나나

문서3 : 길고 노란 바나나 바나나

문서4 : 저는 과일이 좋아요

# TF-IDF(Term Frequency-Inverse Document Frequency)

---

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

>  $tf(d,t)$  : 특정 문서  $d$ 에서의 특정 단어  $t$ 의 등장 횟수.

>  $df(t)$  : 특정 단어  $t$ 가 등장한 문서의 수.

>  **$idf(d, t)$  :  $df(t)$ 에 반비례하는 수.**

문서1 : 먹고 싶은 사과

문서2 : 먹고 싶은 바나나

문서3 : 길고 노란 바나나 바나나

문서4 : 저는 과일이 좋아요

# TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

> **tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.**

> df(t) : 특정 단어 t가 등장한 문서의 수.

> idf(d, t) : df(t)에 반비례하는 수.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	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

문서1 : 먹고 싶은 사과

문서2 : 먹고 싶은 바나나

문서3 : 길고 노란 바나나 바나나

문서4 : 저는 과일이 좋아요

# TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

>  $tf(d,t)$  : 특정 문서 d에서의 특정 단어 t의 등장 횟수.

>  **$df(t)$  : 특정 단어 t가 등장한 문서의 수.**

>  $idf(d, t)$  :  $df(t)$ 에 반비례하는 수.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	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

현재 바나나의 df의 값은?



# TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

>  $tf(d,t)$  : 특정 문서 d에서의 특정 단어 t의 등장 횟수.

>  **$df(t)$  : 특정 단어 t가 등장한 문서의 수.**

>  $idf(d, t)$  :  $df(t)$ 에 반비례하는 수.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	200	0	1	0	0
문서3	0	1	1	0	100	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

현재 바나나의 df의 값은?

# TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

>  $tf(d,t)$  : 특정 문서 d에서의 특정 단어 t의 등장 횟수.

>  $df(t)$  : 특정 단어 t가 등장한 문서의 수.

>  **$idf(d, t)$  :  $df(t)$ 에 반비례하는 수.**

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	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

그렇다면 바나나의 idf는?  
idf는 df의 역수로 하여  $1/2$ ? No.

# TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

> tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.

> df(t) : 특정 단어 t가 등장한 문서의 수.

> **idf(d, t) : df(t)에 반비례하는 수.**

idf는 df에 반비례하도록 설계한 후, log를 씌운다.

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

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	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

# TF-IDF(Term Frequency-Inverse Document Frequency)

> idf(d, t)에는 왜 log를 씌울까?

> log의 밑은 10을 사용한다고 가정하고, 단어의 df에 따른 id 값의 변화를 보자.

$$idf(d, t) = \log(n/df(t))$$

$$n = 1,000,000$$

단어 $t$	$df(t)$	$idf(d, t)$
word1	1	6
word2	100	4
word3	1,000	3
word4	10,000	2
word5	100,000	1
word6	1,000,000	0

로그 사용

$$idf(d, t) = n/df(t)$$

$$n = 1,000,000$$

단어 $t$	$df(t)$	$idf(d, t)$
word1	1	1,000,000
word2	100	10,000
word3	1,000	1,000
word4	10,000	100
word5	100,000	10
word6	1,000,000	1

로그 미사용



# TF-IDF(Term Frequency-Inverse Document Frequency)

> 로그를 사용하지 않으면 idf의 값이 기하급수적으로 커질 수 있다.

$$idf(d, t) = \log(n/df(t))$$

$$n = 1,000,000$$

단어 $t$	$df(t)$	$idf(d, t)$
word1	1	6
word2	100	4
word3	1,000	3
word4	10,000	2
word5	100,000	1
word6	1,000,000	0

로그 사용

$$idf(d, t) = n/df(t)$$

$$n = 1,000,000$$

단어 $t$	$df(t)$	$idf(d, t)$
word1	1	1,000,000
word2	100	10,000
word3	1,000	1,000
word4	10,000	100
word5	100,000	10
word6	1,000,000	1

로그 미사용

# IDF에 log가 필요한 이유

---

- TF-IDF는 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단하며, 특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단한다.
- 불용어 등과 같이 자주 쓰이는 단어들은 비교적 자주 쓰이지 않는 단어들보다 최소 수십배 자주 등장한다.
- 비교적 자주 쓰이지 않는 단어들조차 희귀 단어들과 비교하면 최소 수백 배는 더 자주 등장하는 편이다.
- log를 씌워주지 않으면, 희귀 단어들에 엄청난 가중치가 부여될 수 있다.

로그를 씌우면 이런 격차를 줄이는 효과가 있다.

# TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d, t)$  : 특정 문서 d에서의 특정 단어 t의 등장 횟수

## DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	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

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

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	$\ln(4/(1+1)) = 0.693147$
싫은	$\ln(4/(2+1)) = 0.287682$
저는	$\ln(4/(1+1)) = 0.693147$
좋아요	$\ln(4/(1+1)) = 0.693147$

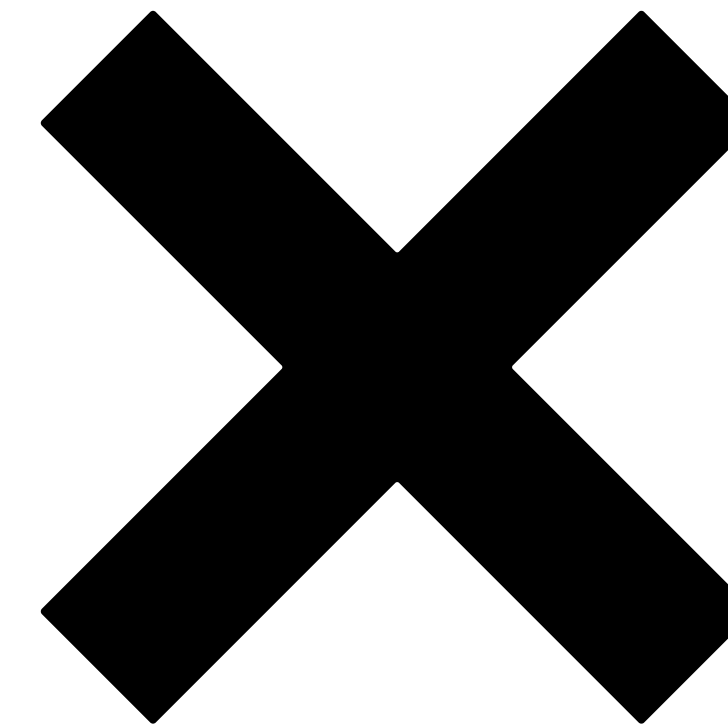


# TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d, t)$  : 특정 문서 d에서의 특정 단어 t의 등장 횟수

DTM

-	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



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

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	$\ln(4/(1+1)) = 0.693147$

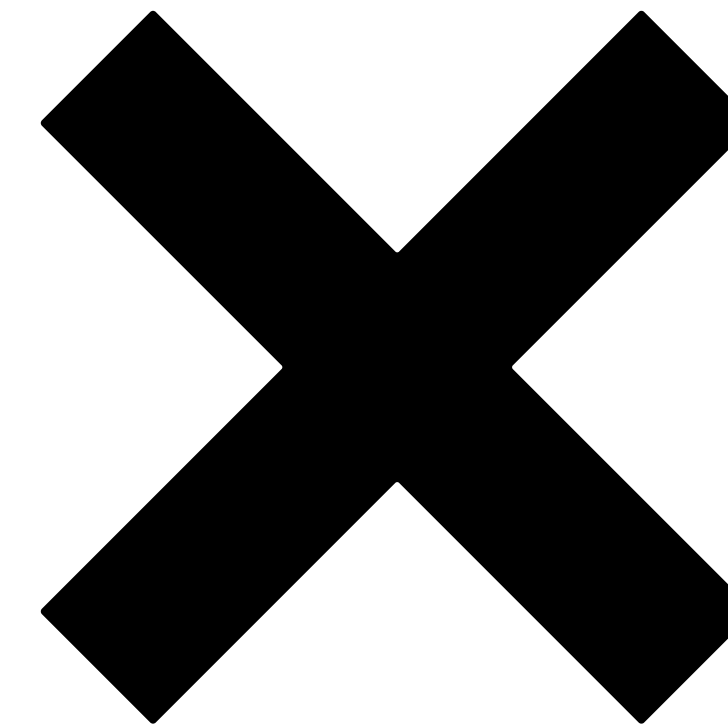


# TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d, t)$  : 특정 문서 d에서의 특정 단어 t의 등장 횟수

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	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



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

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	$\ln(4/(1+1)) = 0.693147$
싫은	$\ln(4/(2+1)) = 0.287682$
저는	$\ln(4/(1+1)) = 0.693147$
좋아요	$\ln(4/(1+1)) = 0.693147$

IDF 값을 보면 2회 등장한 단어들이 값이 더 낮다.

# TF-IDF(Term Frequency-Inverse Document Frequency)

>  $tf(d, t)$  : 특정 문서  $d$ 에서의 특정 단어  $t$ 의 등장 횟수.

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

-	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

같은 단어라도 TF-IDF값은 다르다.  
이는 해당 문서의 TF값에 영향을 받기 때문이다.

# TF-IDF(Term Frequency-Inverse Document Frequency)

>  $tf(d, t)$  : 특정 문서  $d$ 에서의 특정 단어  $t$ 의 등장 횟수.

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

-	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-IDF는 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단하며, 특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단한다.



# TF-IDF(Term Frequency-Inverse Document Frequency)

---

- TF-IDF는 여전히 현업에서도 굉장히 많이 쓰이는 벡터화 방법.
- 문서를 벡터화한다면 각 문서 간의 유사도를 구할 수 있다.
- 그리고 문서 간 유사도를 구할 수 있다면 이런 것이 가능하다.

1. 문서 클러스터링
2. 유사한 문서 찾기
3. 문서 분류 문제

# TF-IDF(Term Frequency-Inverse Document Frequency)

---

- TF-IDF는 여전히 현업에서도 굉장히 많이 쓰이는 벡터화 방법.
- 문서를 벡터화한다면 각 문서 간의 유사도를 구할 수 있다.
- 그리고 문서 간 유사도를 구할 수 있다면 이런 것이 가능하다.

1. 문서 클러스터링
2. 유사한 문서 찾기
3. 문서 분류 문제

+ 물론, 인공 신경망으로 문서 임베딩 벡터를 얻는 방법도 존재.  
Ex) Doc2Vec, Sent2Vec, Universal Sentence Encoder, ELMO, SBERT 등 ..



1. 문서가 있을 때, 이를 DTM으로 표현한다면 문서 하나가 벡터가 된다.
2. 문서가 있을 때, 문서 내의 모든 단어를 워드 임베딩 또는 원-핫 인코딩으로 표현한다면 단어 하나는 벡터가 되고, 문서 자체로 보면 행렬이 된다.
3. 문서의 표현 방법을 신경망으로도 얻을 수 있습니다. Ex) DoC2Vec, ELMo 등...