

NLP WITH PYTORCH

PART I

NLP METHODS & REPRESENTATION

NLP METHODS

3

◆ 자연어 처리 방법들 - 언어 모델 (Language Model, LM)

- ❖ 언어라는 현상을 모델링하고자 단어 **시퀀스(문장)에 확률을 할당(assign)하는 모델**
- ❖ **앞이나 뒤에 있는 단어나 문장 통해 다음에 올 단어나 문장을 예측하는 알고리즘**
- ❖ 수학적으로 이전 단어들을 토대로 다음에 올 단어의 확률을 예측하는 문제
- ❖ 방법
 - 통계 이용한 **통계적 언어 모델(Statistical Language Model)**
 - 기계 학습을 이용한 **인공 신경망 언어 모델(RNN, LSTM...)**

NLP METHODS

4

◆ 자연어 처리 방법들 - 언어 모델 (Language Model, LM)

❖ 단어 시퀀스 확률 할당 → 자연스러운 문장 선택

- 기계 번역(Machine Translation)

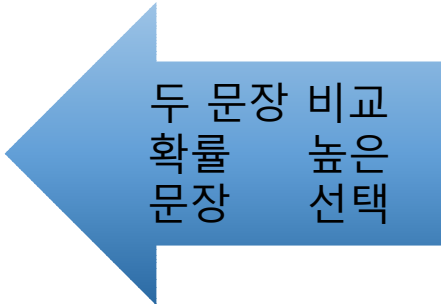
$P(\text{나는 버스를 탔다}) > P(\text{나는 버스를 태운다})$

- 오타 교정(Spell Correction)

선생님이 교실로 부리나케 $P(\text{달려갔다}) > P(\text{갈려갔다})$

- 음성 인식(Speech Recognition)

$P(\text{나는 메롱을 먹는다}) < P(\text{나는 메론을 먹는다.})$



두 문장 비교
확률 높은
문장 선택

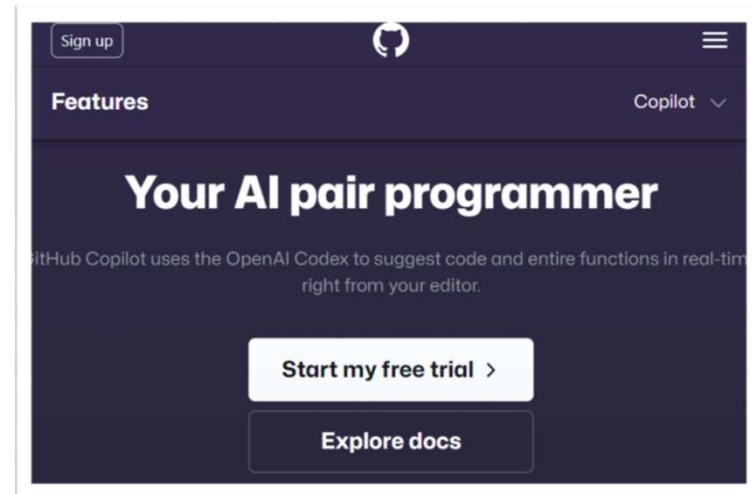
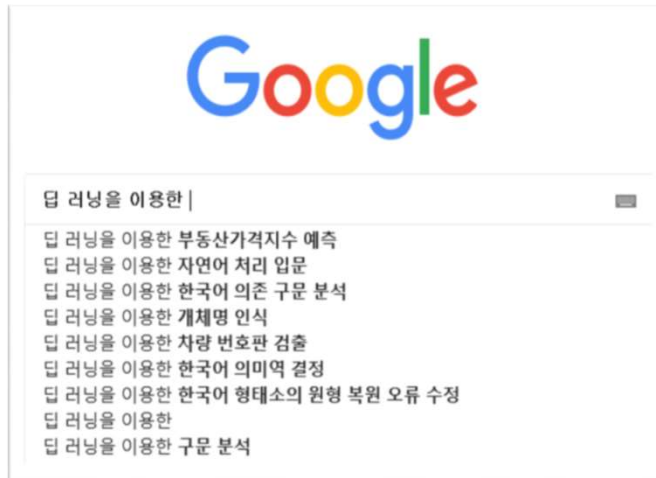
NLP METHODS

5

◆ 자연어 처리 방법들 - 언어 모델 (Language Model, LM)

❖ 자동완성 기능 → 웹 브라우저 검색어 입력

❖ GitHub Copilot → 함수 정의 부분 입력 시 코드 자동 작성



NLP METHODS

6

◆ 자연어 처리 방법들 - 통계적언어모델(SLM:Statistical Language Model)

❖ 전통적인 접근 방법 언어 모델

❖ 조건부 확률 기반

- 각 단어는 문맥이라는 관계로 인해 이전 단어의 영향을 받아 나온 단어
- 모든 단어로부터 하나의 문장이 완성
- 문장의 확률
 - 각 단어들이 이전 단어가 주어졌을 때 다음 단어로 등장할 확률의 곱
 - 카운트 기반으로 다음 단어에 대한 예측 확률 계산
 - 모든 단어의 예측 확률값을 곱함

NLP METHODS

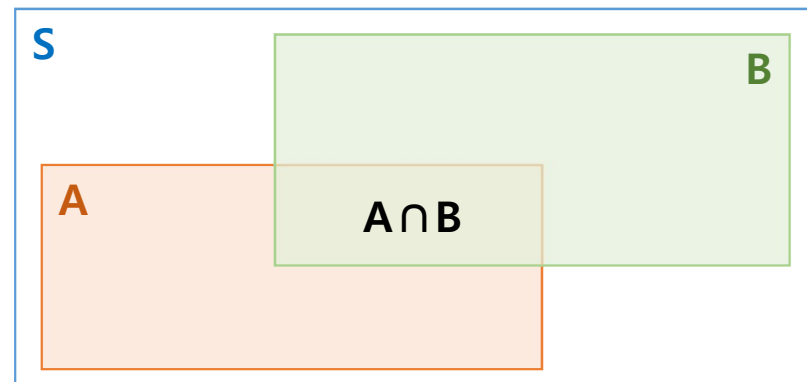
7

◆ 자연어 처리 방법들 - 통계적언어모델(SLM:Statistical Language Model)

❖ 조건부 확률(Conditional probability)

- 어떤 사건 B가 일어났을 때 A가 일어날 확률을 의미
- 어떤 사건이 일어났다는 전제 하에 다른 사건이 일어날 확률
- 사건 B가 일어날 확률에서 사건 A가 동시에 일어날 확률

$$\begin{aligned} P(A|B) &= \frac{P(A \cap B)}{P(B)} = \frac{\frac{n(A \cap B)}{n(S)}}{\frac{n(B)}{n(S)}} \\ &= \frac{n(A \cap B)}{n(B)} \end{aligned}$$



NLP METHODS

8

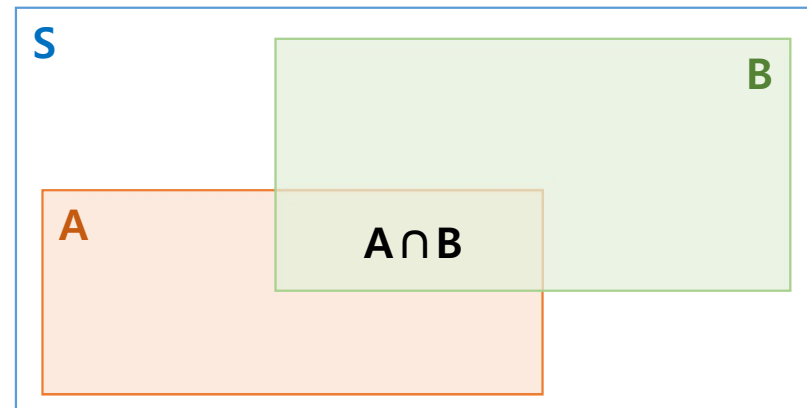
◆ 자연어 처리 방법들 - 통계적언어모델(SLM:Statistical Language Model)

❖ 조건부 확률(Conditional probability)

- 어떤 사건 A가 일어났을 때 B가 일어날 확률 의미

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

$$\begin{aligned} P(A \cap B) &= P(B|A) \cdot P(A) \\ &= P(A|B) \cdot P(B) \end{aligned}$$



NLP METHODS

9

◆ 자연어 처리 방법들 - 통계적언어모델(SLM:Statistical Language Model)

❖ 조건부 확률(Conditional probability)

$S = \{1, 2, 3, 4, 5, 6\}$ $A = \{2, 4, 6\}$ $B = \{3, 6\}$

* A가 일어나고 B가 일어날 확률

$$P(B|A) = P(A \cap B) / P(A) = (1/6) / (3/6) = 1/3$$

$$P(A \cap B) = P(B|A) * P(A)$$

* B가 일어나고 A가 일어날 확률

$$P(A|B) = P(A \cap B) / P(B) = (1/6) / (2/6) = 1/2$$

$$P(A \cap B) = P(A|B) * P(B)$$

NLP METHODS

10

◆ 자연어 처리 방법들 - 통계적언어모델(SLM:Statistical Language Model)

❖ 조건부 확률(Conditional probability)

3개의 조건부 확률 관계

$$\begin{aligned}P(A \cap B \cap C) &= P(C \cap A \cap B) \\&= P(C | A \cap B) * P(A \cap B) \\&= P(C | A \cap B) * P(B|A) * P(A) \\&= P(A) * P(B|A) * P(C | A \cap B)\end{aligned}$$

4개의 조건부 확률 관계

$$\begin{aligned}P(A \cap B \cap C \cap D) &= P(D \cap C \cap A \cap B) \\&= P(D|C \cap A \cap B) * P(C | A \cap B) * P(A \cap B) \\&= P(D|C \cap A \cap B) * P(C | A \cap B) * P(B | A) * P(A) \\&= P(A) * P(B|A) * P(C | A \cap B) * P(D | A \cap B \cap C)\end{aligned}$$

연쇄 법칙(Chain rule) : 일반화

$$P(A_1 \cap A_2 \cap \dots \cap A_K) = P(A_1) * P(A_2|A_1) * P(A_3|A_2, A_1) \dots * P(A_K | A_1, A_2, \dots, A_{K-1})$$

NLP METHODS

11

◆ 자연어 처리 방법들 - 통계적언어모델(SLM:Statistical Language Model)

❖ 조건부 확률(Conditional probability)

문장 'An adorable little boy is spreading smiles'의 확률 → 각 단어에 대한 확률들 곱

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

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

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

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

NLP METHODS

12

◆ 자연어 처리 방법들 - 통계적언어모델(SLM:Statistical Language Model)

❖ 조건부 확률(Conditional probability) 한계

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{\text{Count}(\text{its water is so transparent that the})}{\text{Count}(\text{its water is so transparent that})}$$

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{500}{10000} = 0.05 (5\%)$$

[한계] its water is so transparent that 단어 시퀀스가 없는 경우, 분모 0 → 확률 정의 X

충분한 데이터 관측하지 못해 언어를 정확히 모델링하지 못하는 문제, 희소 문제(sparsity problem)

NLP METHODS

13

◆ 자연어 처리 방법들 - 통계적언어모델(SLM:Statistical Language Model)

❖ 한계 → 희소(Sparsity) 문제

- 단어의 조합에 대한 경우의 수가 엄청나게 많음
- 아무리 많은 데이터를 수집하더라도 특정 단어 조합의 경우 Dataset에 한번도 존재하지 않을 수 있음



❖ 해결책

- n-gram 모델 적용 → 완벽한 해결책은 안됨 !!!

NLP METHODS

14

◆ 자연어 처리 방법들 - N-gram 언어모델(N-gram Language Model)

❖ N-gram이란?

- 전체 문장에서 단어를 **N개수 만큼 묶은 것**
- 카운트에 기반한 통계적 접근 사용
- **N개의 단어만 고려하여 판단 즉, 일부 단어만 고려하는 접근 방법을 사용함**

NLP METHODS

15

◆ 자연어 처리 방법들 - N-gram 언어모델(N-gram Language Model)

❖ N-gram이란?

Uni-Gram

This	is	Big	Data	AI	Book
------	----	-----	------	----	------

 : 모든 단어 완전히 서로 독립

Bi-Gram

This is	is Big	Big Data	Data AI	AI Book
---------	--------	----------	---------	---------

 : 바로 전 단어에만 의존

Tri-Gram

This is Big	is Big Data	Big Data AI	Data AI Book
-------------	-------------	-------------	--------------

 : 이전 2개 단어에만 의존

NLP METHODS

16

◆ 자연어 처리 방법들 - N-gram 언어모델(N-gram Language Model)

❖ N-gram이란?

- 이전 K개의 단어 묶음에 기반해서 다음에 올 단어를 예측하는 언어 모델
- $N=K+1$
 - $N=2$ 이면 이전에 존재하는 1개 단어
 - $N=3$ 이면 이전에 존재하는 2개 단어

예) 7-gram 예시 → 이전 단어 6개로 현재 단어 다음에 올 단어 예측
P(the | its water is so transparent that)

이전 단어 6개

NLP METHODS

17

◆ 자연어 처리 방법들 - N-gram 언어모델(N-gram Language Model)

❖ N-gram이란?

- 문제점

- 단어의 조합에 대한 경우의 수가 엄청나게 많기 때문에 아무리 많은 데이터를 수집하더라도
- 특정 단어 조합의 경우 Dataset에 한번도 존재하지 않을 수 있음

- 해결방안

- 딥러닝 기반 언어 모델 주로 사용

TEXT REPRESENTATION

18

◆ 텍스트 수치 표현 방법들

▪ 국소/이산 표현 (Local/Discrete)

- puppy, cute, lovely 각 단어에 1번, 2번, 3번 숫자 맵핑(mapping) 부여
- 단어 하나의 의미만 참고
- 종류 : One-Hot Vector, N-gram, 카운트 기반 (BOW/DTM)

▪ 분산/연속 표현 (Distributed/Continuous)

- 해당 단어 표현 위해 주변 단어 참고
- 단어와 여러 의미 관계를 함께 계산하여 다차원에 표현
- 종류 : 예측기반 (Word2Vec, Doc2Vec), 카운트 기반(LSA, Glove)

TEXT REPRESENTATION

19

◆ 텍스트 수치 표현 방법들

▪ 희소 표현 (Sparse)

- "값이 0인 원소가 많은 벡터" 이르는 말로 유효한 값이 적다는 의미
- 사용 단어집합의 크기만큼 벡터의 차원이 정해지며 유효한 값 이외에는 '0' → 고차원
- 종류 : 원 핫 인코딩(One-Hot Encoding), 정수 인코딩

▪ 밀집 표현 (Dense)

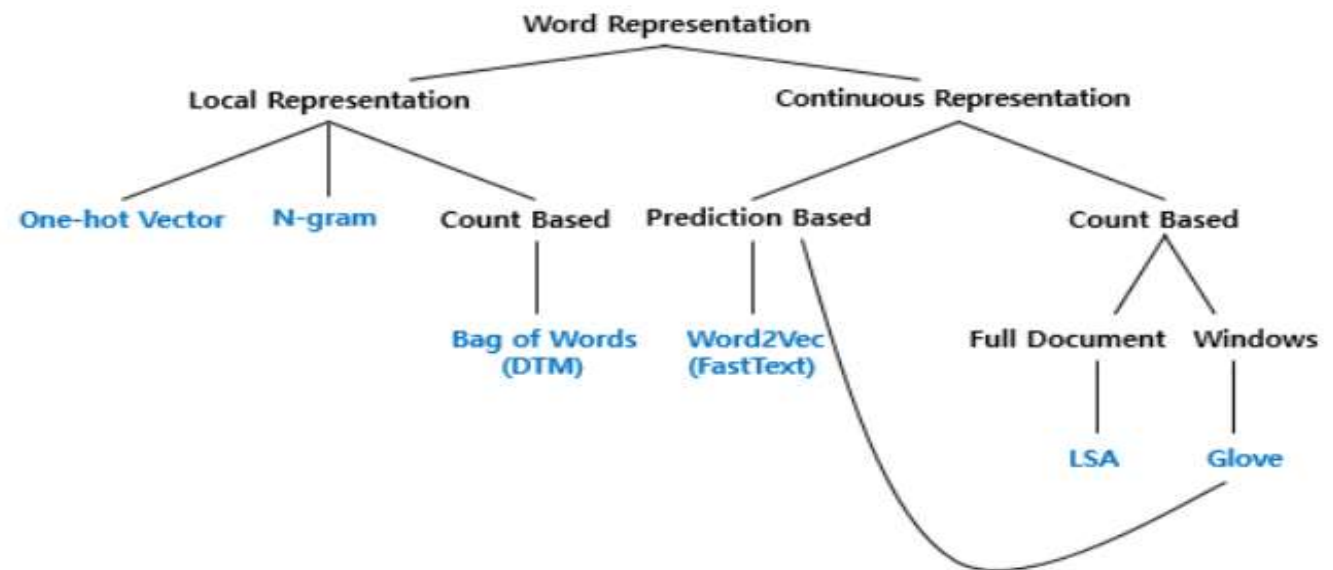
- 희소 표현의 반대 표현
- 단어의 의미를 크기가 정해진 저차원 벡터에 실수값으로 표현
- 실수로 조정된 값을 저차원에 뽁뽁히 모아서 표현 → '밀집표현'

TEXT REPRESENTATION

20

◆ 텍스트 수치 표현 방법들

❖ 단어표현 카테고리화



TEXT REPRESENTATION

21

◆ 텍스트 수치 표현 방법들 - BOW(Bag Of Words)

- 정보 검색과 텍스트 마이닝 분야에서 주로 사용
- 수치화 표현 후 통계적 접근 방법 통해 사용
- 단어의 등장 순서 고려 하지 않고 출현 빈도수 카운트(Count)하여 단어를 수치화
- 문맥 의미 반영 X, 희소 행렬 문제
- 방법
 - ① 여러 문장들의 단어 리스트 생성 → 단어 집합 생성
 - ② 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터 생성

TEXT REPRESENTATION

22

◆ 텍스트 수치 표현 방법들 - BOW(Bag Of Words)

■ 예시

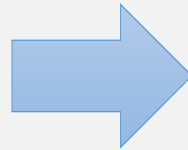
단어 정수 인덱스 할당 =====>

- ① 오늘은 수요일 입니다.
- ② 오늘은 어제보다 덥고 오늘은 수요일 입니다.
- ③ 가을 입니다.

오늘은 1	수요일 2	입니다 3	어제보다 4	덥고 5	가을 6
1	1	1	0	0	0
2	1	1	1	1	0
0	0	1	0	0	1

문장 1과 문장 2 비교 => O O O X X O

문장 1과 문장 3 비교 => X X O O O X



문장 1과 문장 2가 유사합니다!

TEXT REPRESENTATION

23

◆ 텍스트 수치 표현 방법들 - BOW(Bag Of Words)

❖ sklearn의 CountVectorizer

- 단어의 빈도를 Count하여 Vector 생성
- 영어에 대해 손쉽게 BoW 생성
- 기본적으로 길이가 2이상인 문자에 대해서만 토큰으로 인식
- 단지 띄어쓰기만을 기준으로 단어를 자르는 낮은 수준의 토큰화를 진행

TEXT REPRESENTATION

24

◆ 텍스트 수치 표현 방법들 - BOW(Bag Of Words)

❖ sklearn의 CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
corpus = ['you know I want your love. because I love you.']  
vector = CountVectorizer()
```

```
vector.fit(corpus) # 단어 사전 생성  
print(f[vocabulary_ 단어 인덱스] => {vector.vocabulary_})
```

```
result=vector.transform(corpus).toarray() # 단어별 빈도수  
print(f[문장 단어 빈도수] {corpus }=> {result})
```


TEXT REPRESENTATION

25

◆ 텍스트 수치 표현 방법들 - BOW(Bag Of Words)

❖ sklearn의 CountVectorizer

- 사용자 지정 불용어

```
corpus = ['you know I want your love. because I love you.']
```

```
vector = CountVectorizer( stop_words=['the', 'I'] ) # 불용어 지정
```

TEXT REPRESENTATION

26

◆ 텍스트 수치 표현 방법들 - BOW(Bag Of Words)

❖ sklearn의 CountVectorizer

- CountVectorizer 내부 불용어 적용

```
corpus = ['you know I want your love. because I love you.']
```

```
vector = CountVectorizer( stop_words='english') # 내부 불용어 지정
```

TEXT REPRESENTATION

27

◆ 텍스트 수치 표현 방법들 - DTM(Document-Term Matrix)

- 서로 다른 문서들의 BoW들을 결합한 표현 방법인 문서 단어 행렬(DTM) 표현
- 다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것
- 각 문서에 대한 BoW를 하나의 행렬로 만든 것
- 방법
 - 각 문서의 단어 벡터 생성
 - 해당 단어가 존재하는 빈도 수 채움

TEXT REPRESENTATION

28

◆ 텍스트 수치 표현 방법들 - DTM(Document-Term Matrix)

- 문서1 : 먹고 싶은 사과
- 문서2 : 먹고 싶은 바나나
- 문서3 : 길고 노란 바나나 바나나
- 문서4 : 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

TEXT REPRESENTATION

29

◆ 텍스트 수치 표현 방법들 - DTM(Document-Term Matrix)

❖ 한계

- 각 문서에는 중요한 단어와 불필요한 단어 혼재
- 빈도가 높아도 의미가 없는 단어들 존재
- 단어가 많을 수록 벡터 사이즈 커짐 → 메모리 낭비
- 대부분이 0인 값으로 희소 벡터/희소 행렬 → 희소 표현

TEXT REPRESENTATION

30

◆ 텍스트 수치 표현 방법들 - TF-IDF

- Term Frequency-Inverse Document Frequency 약자
- 단어의 빈도와 역 문서 빈도(문서의 빈도에 특정 식을 취함)를 사용하여 DTM 내각 **단어들마다 중요한 정도를 가중치로 주는 방법**
- 기존의 DTM을 사용하는 것보다 **보다 많은 정보를 고려하여 문서들을 비교 가능**
- 방법
 - DTM 생성
 - 가중치 부여

TEXT REPRESENTATION

31

◆ 텍스트 수치 표현 방법들 - TF-IDF

- 특정 단어가 특정 문서에만 자주 사용 → 개별 문서 특징 정보가 됨!
- 특정 단어가 매우 많은 문서에 자주 사용 → 개별 문서 특징 정보 아님
 - TF(Term Frequency) : 문서에 얼마나 자주 나타났는지 지표
 - DF(Document Frequency) : 해당 단어가 몇 개의 문서에서 나타났는지 지표
 - IDF(Inverse Document Frequency) : 문서수/DF
- TF ▲ DF ▼ ← 중요 단어임
- TF ▲ DF ▲ ← 중요 단어 아님

TEXT REPRESENTATION

32

◆ 텍스트 수치 표현 방법들 - TF-IDF

■ 수치 표현

-	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

-	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

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

TEXT REPRESENTATION

33

◆ 텍스트 수치 표현 방법들 - TF-IDF(Term Frequency-Inverse Document Frequency)

```
import pandas as pd
from math import log

docs = [ '먹고 싶은 사과', '먹고 싶은 바나나',
         '길고 노란 바나나 바나나', '저는 과일이 좋아요' ]

vocab = list(set(w for doc in docs for w in doc.split()))
vocab.sort()
```

TEXT REPRESENTATION

34

◆ 텍스트 수치 표현 방법들 - TF-IDF(Term Frequency-Inverse Document Frequency)

```
# 총 문서의 수
```

```
N = len(docs)
```

```
# 문서에 나타난 단어 빈도 수
```

```
def tf(t, d):
```

```
    return d.count(t)
```

```
# 총 문서에서 단어가 나타난 빈도
```

```
def idf(t):
```

```
    df = 0
```

```
    for doc in docs : df += 1 if t in doc
```

```
    return log( N/(df+1) )
```

```
# TF-IDF 계산
```

```
def tfidf(t, d): return tf(t,d)* idf(t)
```

TEXT REPRESENTATION

35

◆ 텍스트 수치 표현 방법들 - TF-IDF(Term Frequency-Inverse Document Frequency)

```
result = []  
# 각 문서에 대해서 아래 연산 반복  
for i in range(N):  
    result.append([])  
    d = docs[i]  
    for j in range(len(vocab)):  
        t = vocab[j]  
        result[-1].append(tf(t, d))
```

```
tf_ = pd.DataFrame(result,  
                    columns = vocab)
```

TEXT REPRESENTATION

36

◆ 텍스트 수치 표현 방법들 - TF-IDF(Term Frequency-Inverse Document Frequency)

▪ TF

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
0	0	0	0	1	0	1	1	0	0
1	0	0	0	1	1	0	1	0	0
2	0	1	1	0	2	0	0	0	0
3	1	0	0	0	0	0	0	1	1

TEXT REPRESENTATION

37

◆ 텍스트 수치 표현 방법들 - TF-IDF(Term Frequency-Inverse Document Frequency)

```
result = []  
for j in range(len(vocab)):  
    t = vocab[j]  
    result.append(idf(t))
```

```
idf_ = pd.DataFrame(result, index=vocab, columns=["IDF"])
```

TEXT REPRESENTATION

38

◆ 텍스트 수치 표현 방법들 - TF-IDF(Term Frequency-Inverse Document Frequency)

▪ IDF

	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
0	0	0	0	1	0	1	1	0	0
1	0	0	0	1	1	0	1	0	0
2	0	1	1	0	2	0	0	0	0
3	1	0	0	0	0	0	0	1	1



	IDF
과일이	0.693147
길고	0.693147
노란	0.693147
먹고	0.287682
바나나	0.287682
사과	0.693147
싫은	0.287682
저는	0.693147
좋아요	0.693147

TEXT REPRESENTATION

39

◆ 텍스트 수치 표현 방법들 - TF-IDF(Term Frequency-Inverse Document Frequency)

```
result = []

for i in range(N):
    result.append([])
    d = docs[i]
    for j in range(len(vocab)):
        t = vocab[j]
        result[-1].append(tfidf(t,d))
```

```
tfidf_ = pd.DataFrame(result,
                        columns = vocab)
```

40

- TF-IDF

[illegible]

TEXT REPRESENTATION

41

◆ 텍스트 수치 표현 방법들 - TF-IDF(Term Frequency-Inverse Document Frequency)

▪ sklearn. TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [ 'you know I want your love', 'I like you', 'what should I do ' ]

tfidf = TfidfVectorizer().fit(corpus)
print( tfidf.transform(corpus).toarray() )
print( tfidf.vocabulary_ )
```

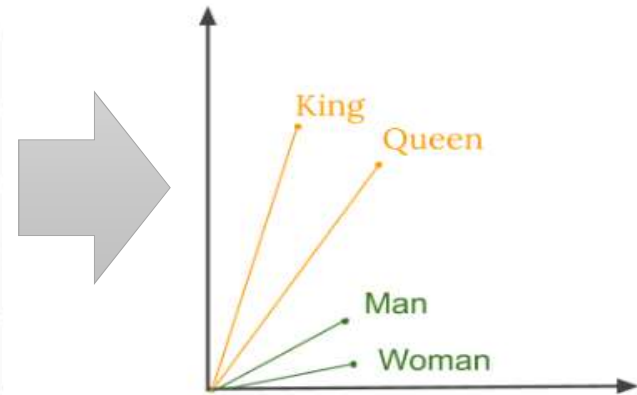
TEXT REPRESENTATION

42

◆ 텍스트 수치 표현 방법들 - 벡터 유사도(Vector Similarity)

- 개별 단어를 문맥을 가진 N차원 공간에 벡터로 표현
- 같은 의미를 가진 단어가 비슷한 표현을 갖는 n차원 공간 위치

	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



TEXT REPRESENTATION

43

◆ 텍스트 수치 표현 방법들 - 벡터 유사도(Vector Similarity)

- 기존의 카운트 기반, TF-IDF기반의 문제점 해결 위한 표현 방법
- 문장이나 문서의 비슷한 의미의 단어를 모아서 표현
- 문서의 단어들의 비슷한 정도를 기준으로 수치화
- 방법
 - 코사인 유사도 (Cosine Similarity)
 - 유클리드 거리(Euclidean distance)
 - 자카드 유사도(Jaccard similarity)

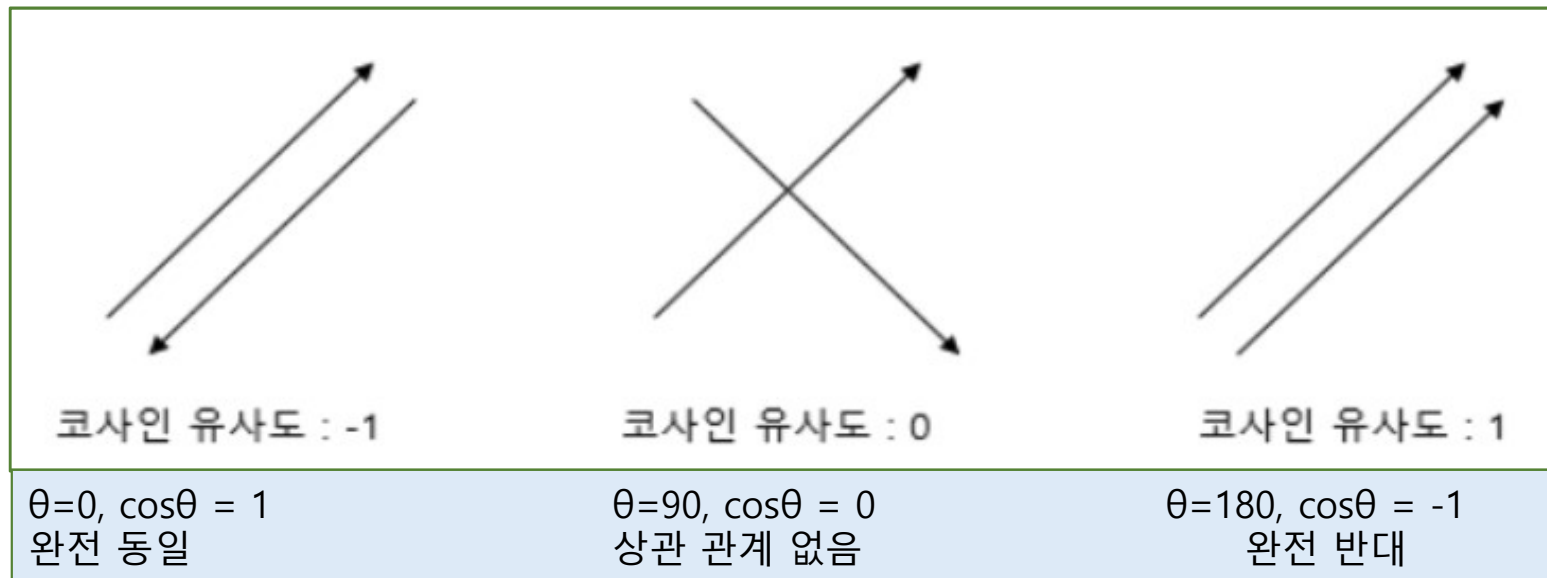
TEXT REPRESENTATION

44

◆ 텍스트 수치 표현 방법들 - 벡터 유사도(Vector Similarity)

❖ 코사인 유사도 (Cosine Similarity)

- 두 벡터 간의 코사인 각도 이용하여 구할 수 있는 두 벡터의 유사도 의미



TEXT REPRESENTATION

45

◆ 텍스트 수치 표현 방법들 - 벡터 유사도(Vector Similarity)

❖ 코사인 유사도 (Cosine Similarity)

```
from sklearn.feature_extraction.text import TfidfVectorizer

doc_list=['if you take the blue pill, the story ends' ,
          'if you take the red pill, you stay in Wonderland',
          'if you take the red pill, I show you how deep the rabbit hole goes']

tfidf_vect = TfidfVectorizer()
feature_vect = tfidf_vect.fit_transform( doc_list )
```

TEXT REPRESENTATION

46

◆ 텍스트 수치 표현 방법들 - 벡터 유사도(Vector Similarity)

❖ 코사인 유사도 (Cosine Similarity)

```
from sklearn.metrics.pairwise import cosine_similarity

## cosine_similarity(X, y)
# - X : 비교 기준이 되는 문서의 피처 행렬
# - y : 비교하고자 하는 문서의 피처 행렬
result = cosine_similarity(feature_vect[0] , feature_vect)
print( result )
```

TEXT REPRESENTATION

47

◆ 패딩(Padding)

- 가변 길이의 문장들을 동일 길이로 맞추어 주는 것
- 지정된 길이보다 길면 자르기
- 지정된 길이보다 짧으면 0으로 채우기

TEXT REPRESENTATION

48

◆ 임베딩(Embedding)

❖ One-Hot-Encoding 표현 문제점

0	0	1	0	0	0	0	---	---	0
---	---	---	---	---	---	---	-----	-----	---

- 데이터 표현 형태가 희소(sparse)해진다!
- 단어집합 size에 비례해서 차원이 커짐

TEXT REPRESENTATION

49

◆ 임베딩(Embedding)

- One-Hot-Encoding 표현 문제점 해결 방안
- Sparse → Dense한 표현으로 변환

$W_{\text{embedding}} * X_{\text{one-hot}} = X_{\text{embedding}}$
원본 데이터 x 임베딩 행렬 → 변환

** 임베딩 벡터

- 어떤 모델에 어떤 데이터셋으로 학습을 시켜 놓은 임베딩 벡터
- text2Vecotr, word2Vetor 등등

Ont-hot vector	Embedding vector
[1 0 0 0 0]	[0.1 4.2 1.5 1.1 2.8]
[0 1 0 0 0]	[1.0 3.1 2.5 0.7 1.1]
[0 0 1 0 0]	[0.3 2.1 1.5 2.1 0.1]
[0 0 0 1 0]	[2.2 1.4 0.5 0.9 1.1]
[0 0 0 0 1]	[0.7 1.7 0.5 0.3 0.2]

TEXT REPRESENTATION

50

◆ 임베딩(Embedding)

❖ 장점

- 차원 축소 효과
- 희박한 데이터 표현 형태를 **밀집한 데이터 표현 형태**로 학습 **성능 향상**
- 유사한 의미 단어를 **유사한 벡터 표현**을 가짐으로 연관관계 정보 담김
- 유사한 의미 단어 벡터들 사이 사칙연산 가능
 - <https://word2vec.kr/search/>
 - tSEN이용해서 좌표평면에 유사 의미 단어 위치 확인 가능

TEXT REPRESENTATION

51

◆ 자연어 처리 라이브러리/패키지

- Hugging Face 라이브러리
 - 복잡한 Transformer 계열의 복잡하고 큰 모델을 손쉽게 구현 도와주는 라이브러리



**The AI community
building the future.**

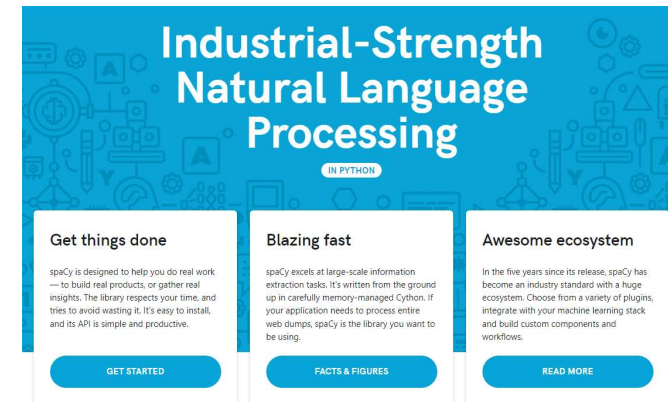
Build, train and deploy state of the art models powered by
the reference open source in machine learning.

TEXT REPRESENTATION

52

◆ 자연어 처리 라이브러리/패키지

- spaCy 라이브러리
 - 뛰어난 수행 능력으로 최근 가장 주목 받는 NLP
 - 고급 자연어 처리 라이브러리 , 한국어 지원 하지 않음



PART I

TEXT PREPROCESSING PROGRAMMING

TEXT PREPROCESSING

54

◆ 전처리 단계 - 정제(Cleaning)

❖ 정규 표현식(Regular Expression) : 텍스트 전처리에 아주 유용한 도구

특수 문자	설명
.	한 개의 임의의 문자 나타냄. (줄바꿈 문자인 $\backslash n$ 는 제외)
?	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있음. (문자가 0개 또는 1개)
*	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있음. (문자가 0개 이상)
+	앞의 문자가 최소 한 개 이상 존재. (문자가 1개 이상)
^	뒤의 문자열로 문자열이 시작.
\$	앞의 문자열로 문자열이 끝남.

TEXT PREPROCESSING

55

◆ 전처리 단계 - 정제(Cleaning)

❖ 정규 표현식(Regular Expression) : 텍스트 전처리에 아주 유용한 도구

특수 문자	설명
{숫자}	숫자만큼 반복
{숫자1, 숫자2}	숫자1 이상 숫자2 이하만큼 반복. ?, *, +를 이것으로 대체 가능
{숫자,}	숫자 이상만큼 반복
[]	대괄호 안의 문자들 중 한 개의 문자와 매치 범위를 지정 가능 (예: [a-zA-Z]는 알파벳 전체 의미)
[^문자]	해당 문자를 제외한 문자를 매치
	A B와 같이 쓰이며 A 또는 B의 의미

TEXT PREPROCESSING

56

◆ 전처리 단계 - 정제(Cleaning)

❖ 정규 표현식(Regular Expression) : 텍스트 전처리에 아주 유용한 도구

문자 규칙	설명
WWW	역 슬래쉬 문자 자체 의미
Wwd	모든 숫자 의미. [0-9]와 의미 동일
WWD	숫자 제외한 모든 문자 의미. [^0-9]와 의미가 동일
Wws	공백 의미. [WtWnWrWfWv]와 의미가 동일
WWS	공백 제외한 문자 의미 [^ WtWnWrWfWv]와 의미가 동일
WWw	문자 또는 숫자 의미 [a-zA-Z0-9]와 의미가 동일
WWW	문자 또는 숫자가 아닌 문자 의미 [^a-zA-Z0-9]와 의미가 동

TEXT PREPROCESSING

57

◆ 전처리 단계 - 정제(Cleaning)

❖ 정규 표현식(Regular Expression) : 텍스트 전처리에 아주 유용한 도구

re.compile()	정규표현식 컴파일 함수
re.search()	문자열 전체에 대해서 정규표현식과 매치되는지 검색
re.match()	문자열 처음부분 정규표현식과 매치되는지 검색
re.split()	정규 표현식을 기준으로 문자열을 분리하여 리스트로 리턴
re.findall()	정규 표현식과 매치되는 모든 경우의 문자열 찾아 리스트로 리턴
re.finditer()	정규 표현식과 매치되는 모든 경우의 문자열에 대한 이터레이터 객체 리턴
re.sub()	정규 표현식과 일치하는 부분에 대해서 다른 문자열 대체

TEXT PREPROCESSING

58

◆ 전처리 단계 - 정제(Cleaning)

❖ 정규 표현식(Regular Expression) : 텍스트 전처리에 아주 유용한 도구

```
import re

r = re.compile("a.c") # a와 c 사이 어떤 1개 문자

ret1=r.search("kkk")
ret2=r.search('abc')

print(f'fret1=> {ret1} ret1=> {ret2}')
```

TEXT PREPROCESSING

59

◆ 전처리 단계 - 정제(Cleaning)

❖ 정규 표현식(Regular Expression) : 토큰화

```
from nltk.tokenize import RegexpTokenizer
```

```
text = "Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery  
goes for a pastry shop"
```

```
tokenizer1 = RegexpTokenizer("[Ww]+")
```

```
tokenizer2 = RegexpTokenizer("Ws+", gaps=True)
```

```
print(tokenizer1.tokenize(text))
```

```
print(tokenizer2.tokenize(text))
```