2. 텍스트 전처리

언어의 유형

유형	언어	특징
굴절어	라틴어, 독일어, 러시아어, 힌디어, 프랑스어	단어의 형태가 변하면서 문법적 기능이 정해짐
고립어	현대 영어, 중국어, 베트남어, 태국어	어순에 따라 단어의 문법적 기능이 정해짐
교착어	한국어, 일본어, 몽골어, 터키어, 헝가리어, 핀란드어	어간에 접사가 붙어 의미와 문법적 기능이 정해짐

한국어 동사의 변형 사례

원형	피동	높임	과거	추측	전달	결과
잡						잡다
잡	+히					잡히다
잡	+히	+시				잡히시다
잡	+히	+시	+었			잡히셨다
잡			+았			잡았다
잡				+겠		잡겠다
잡					+더라	잡더라
잡	+히		+었			잡혔다
잡	+히		+었	+겠		잡혔겠다
잡	+히		+었	+겠	+더라	잡혔겠더라
잡			+았	+겠		잡았겠다
…						…
잡	+히	+시	+았	+겠	+더라	잡히시었겠더라

어순이 그다지 중요하지 않음

번호	문장	정상 여부
1	나는 밥을 먹으러 간다	O
2	간다 나는 밥을 먹으러	O
3	먹으러 간다 나는 밥을	O
4	밥을 먹으러 간다 나는	O
5	나는 먹으러 간다 밥을	O
6	나는 간다 밥을 먹으러	O
7	간다 밥을 먹으러 나는	O
8	간다 먹으러 나는 밥을	O
9	먹으러 나는 밥을 간다	X
10	먹으러 밥을 간다 나는	X
11	밥을 간다 나는 먹으러	X
12	밥을 나는 먹으러 간다	O
13	나는 밥을 간다 먹으러	X
14	간다 나는 먹으러 밥을	O
15	먹으러 간다 밥을 나는	O
16	밥을 먹으러 나는 간다	O

한국어의 주요 특징

- 평서문과 의문문의 내용이 같을 수 있음
  - 예: "점심 먹었어."와 "점심 먹었어?"

- 주어가 자주 생략됨
  - 예: "점심 먹었어."와 "점심 먹었어?"

음절, 형태소, 어절, 품사

- **음절(syllable):** 하나의 덩어리로 여겨지는 글자 단위. 초성, 중성, 종성으로 이루어져 있음
- **형태소(morpheme):** 언어에서 의미를 가지는 가장 작은 단위. 형태소를 쪼개면 더 이상 기능이나 의미를 갖지 않음

나는 컴퓨터 공부가 좋아.

실질 형태소: '나', '컴퓨터', '공부', '좋-'
형식 형태소: '는', '가', '아'

음절, 형태소, 어절, 품사

- **어절:** 한 개 이상의 형태소가 모여 구성된 단위. 띄어쓰기 단위와 거의 일치

> 바닷가에﹀왔더니
> 바다와﹀같이﹀당신이﹀생각만﹀나는구려
> 바다와﹀같이﹀당신을﹀사랑하고만﹀싶구려

- **품사(part-of-speech, pos):** 한국어에서는 해당 단어가 수행하는 역할을 기준으로 **체언, 수식언, 관계언, 독립언, 용언**의 5언으로 나눔. 의미에 따라서는 명사, 대명사, 수사, 관형사, 부사, 조사, 감탄사, 동사, 형용사의 9품사로 나눔

교재 2장 내용

- **토큰화:** 문장의 구성 요소인 토큰 단위로 나누는 것. 단어 토큰화와 문장 토큰화가 있음
- **어간 추출과 표제어 추출**
- **불용어(Stopword):** 자주 나오지만 정보 전달력이 약한 단어들을 걸러내는 것
- **정규 표현식(Regular expression):** 불필요한 부분을 분리하는 것
- **원-핫 인코딩(One-hot encoding):** 각 단어에 고유한 단어 벡터를 부여

토큰화(Tokenization)

- 의미있는 단위(단어 또는 문장)로 문장들을 분리하는 것
- 단어 토큰화(Word Tokenization)
  - 문장을 단어 또는 의미있는 문자열로 구분하는 것
  - 구두점(. , ? ! ; 등의 기호)을 포함하는 것이 좋음
  - 영어의 경우 apostrophe가 있는 부분을 처리하는 방식이 경우에 따라 달라질 수 있음
- 단어 토큰화 사례
  - 입력 문장: "It doesn't have a food restaurant."
  - 결과: ['It', 'does', "n't", 'have', 'a', 'food', 'restaurant', '.']
- 문장 토큰화(Sentence Tokenization)
  - 텍스트를 문장별로 나누는 것

nltk를 이용한 영어 토큰화

- 단어 토큰화:

```
from nltk.tokenize import word_tokenize
print(word_tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop."))
```

```
['Do', "n't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr.', 'Jone', "'s", 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
```

- WordPunctTokenizer 단어 토큰화:

```
from nltk.tokenize import WordPunctTokenizer
print(WordPunctTokenizer().tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop."))
```

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

한글 문장 토큰화

- 한글의 경우 단어 토큰화는 잘 사용되지 않음. 문장을 형태소 분석기에 입력하면 품사로 나뉘어진 단어들을 얻을 수 있음
- 문장을 구분하는 경우 KSS(Korean sentence splitter)를 사용할 수 있음
- kss 사용 사례
  - kss를 먼저 설치해야 함

  > pip install kss

  - 다음과 같이 문장 토큰화를 진행

```
import kss
text='딥 러닝 자연어 처리가 재미있기는 합니다. 그런데 문제는 영어보다 한국어로 할 때 너무 어려워요. 농담아니에요. 이제 해보면 알걸요?'
```

```
['딥 러닝 자연어 처리가 재미있기는 합니다.', '그런데 문제는 영어보다 한국어로 할 때 너무 어려워요.', '농담아니에요.', '이제 해보면 알걸요?']
```

nltk를 이용한 영어 품사 태깅(part-of-speech tagging)

- nltk에서는 Penn Treebank POS Tags를 기준으로 태깅을 수행
- 영어 문장에 대해서 토큰화를 한 다음, 품사 태깅을 수행

```
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
text="I am actively looking for Ph.D. students. and you are a
Ph.D. student."
x = word_tokenize(text)
pos_tag(x)
```

```
[('I', 'PRP'), ('am', 'VBP'), ('actively', 'RB'), ('looking',
'VBG'), ('for', 'IN'), ('Ph.D.', 'NNP'), ('students', 'NNS'),
('.', '.'), ('and', 'CC'), ('you', 'PRP'), ('are', 'VBP'),
('a', 'DT'), ('Ph.D.', 'NNP'), ('student', 'NN'), ('.', '.')]
```

koNLPy를 이용한 형태소(morpheme) 분석

- koNLPy는 형태소 분석기를 모은 패키지임: Okt(Open Korea Text), Komoran, Hannanum, Kkma, Mecab 등이 지원됨
- 각 분석기에서 다음 함수들이 지원됨
  - morphs: 품사를 표시하지 않고 형태소 별로 분리함
  - pos: 품사 태깅(part-of-speech tagging)
  - nouns: 명사 추출
- Mecab은 koNLPy에 포함되어 있지 않으므로 별도로 설치해야 함

koNLPy를 이용한 분석 사례

- 분석 사례

```
from konlpy.tag import Kkma
kkma=Kkma()
print(kkma.morphs("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

```
['열심히', '코딩', '하', 'ㄴ', '당신', ',', '연휴', '에', '는', '여행', '을', '가보', '아요']
```

```
print(kkma.pos("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

```
[('열심히','MAG'), ('코딩', 'NNG'), ('하', 'XSV'), ('ㄴ', 'ETD'), ('당신', 'NP'), (',', 'SP'), ('연휴', 'NNG'), ('에', 'JKM'), ('는', 'JX'), ('여행', 'NNG'), ('을', 'JKO'), ('가보', 'VV'), ('아요', 'EFN')]
```

```
print(kkma.nouns("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

```
['코딩', '당신', '연휴', '여행']
```

koNLPy를 이용한 분석 사례

- 문장: "아버지가방에들어가신다"

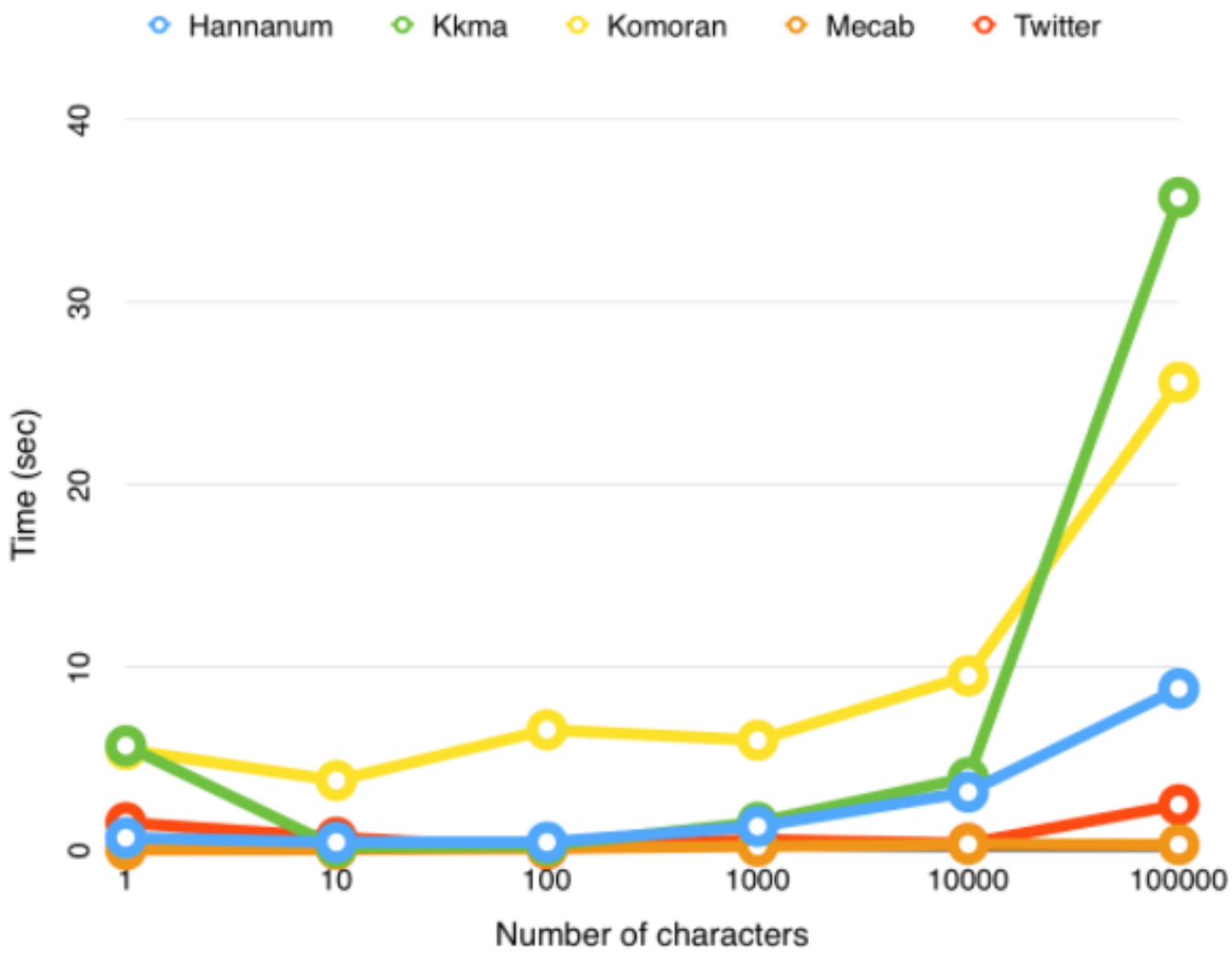
Hannanum	Kkma	Komoran	Mecab	Twitter
아버지가방에 들어가 / N	아버지 / NNG	아버지가방에 들어가신다 / NNP	아버지 / NNG	아버지 / Noun
이 / J	가방 / NNG		가 / JKS	가방 / Noun
시ㄴ다 / E	에 / JKM		방 / NNG	에 / Josa
	들어가 / VV		에 / JKB	들어가신 / Verb
	시 / EPH		들어가 / VV	다 / Eomi
	ㄴ다 / EFN		신다 / EP+EC	

koNLPy를 이용한 분석 사례

- 문장: "아이폰 기다리다 지쳐 애플공홈에서 언락폰질러버렸다"

Hannanum	Kkma	Komoran	Mecab	Twitter
아이폰 / N	아이 / NNG	아이폰 / NNP	아이폰 / NNP	아이폰 / Noun
기다리 / P	폰 / NNG	기다리 / VV	기다리 / VV	기다리 / Verb
다 / E	기다리 / VV	다 / EC	다 / EC	다 / Eomi
지치 / P	다 / ECS	지치 / VV	지쳐 / VV+EC	지쳐 / Verb
어 / E	지치 / VV	어 / EC	애플 / NNP	애플 / Noun
애플공홈 / N	어 / ECS	애플 / NNP	공 / NNG	공홈 / Noun
에서 / J	애플 / NNP	공 / NNG	홈 / NNG	에서 / Josa
언락폰질러버렸다 / N	공 / NNG	홈 / NNG	에서 / JKB	언락폰 / Noun
6+ / N	홈 / NNG	에서 / JKB	언락 / NNG	질 / Verb
128기가실벜 / N	에서 / JKM	언 / NNG	폰 / NNG	러 / Eomi
	언락 / NNG	락 / NNG	질러버렸 / VV+EC+VX+EP	버렸 / Verb
	폰 / NNG	폰 / NNG	다 / EC	다 / Eomi

형태소 분석기 성능


Hannanum
Kkma
Komoran
Mecab
Twitter
Time (sec)
0
10
20
30
40
1
10
100
1000
10000
100000
Number of characters

표제어 추출과 어간 추출

- **표제어 추출(Lemmatization):** 영어에서 am, are, is의 표제어는 be 임
  - nltk에서는 WordNetLemmatizer 함수로 수행
- **어간 추출(Stemming):** nltk에서는 PorterStemmer 함수로 기능을 수행
- **한글의 경우 표제어와 어간을 추출하는 프로그램은 아직 없음**

불용어(Stopword)

- 사용 빈도와 관계없이 문서의 의미 전달 측면에서는 큰 의미가 없는 단어들을 stopword라고 함
- 영어의 경우 nltk에서 불용어 리스트를 제시하고 있음

```
from nltk.corpus import stopwords
stopwords.words('english')[:10]
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
'you', 'your']
```

한글 불용어

- 한글의 경우 불용어를 처리하는 패키지는 없고 공통적인 리스트를 제공하는 사이트들이 있음
- 사례: https://www.ranks.nl/stopwords/korean
- 한글에서는 불용어 리스트를 파일에 만들어 놓고 사용하는 것이 바람직함

정규 표현식(Regular expression)

- 정규 표현식은 특정한 패턴을 찾기 위한 방식임
- 파이썬에서는 re 모듈에서 함수들이 지원되고 있음
- search 함수에 의해 text내에 pattern이 존재하는지 알려줌

```
import re
re.search(patt, text)
```

- 패턴이 존재하지 않으면 응답이 없고, 존재하면 message가 나옴

```
import re
re.search("at", "string data")
```

```
<re.Match object; span=(8, 10), match='at'>
```

```
import re
re.search("at", "this is another string")
```

```
응답이 없음
```

패턴 지정 방법 1

패턴	의미
[ ]	대괄호 안의 문자들 중 한 문자와 매치. [amk]라 하면 a 또는 m 또는 k 중 하나가 존재하면 됨. [a-f]와 같이 범위를 지정할 수 있음. [a-zA-Z]은 알파벳 전체를 의미
[^문자]	해당 문자를 제외한 문자를 매치. [^5-9]는 5~9 숫자가 아니면 매치.
.	한 개의 임의의 문자
?	앞의 문자가 존재할 수도 있고, 존재하지 않을 수 있음(문자가 0개 또는 1개)
*	앞의 문자가 0개 이상 존재
+	앞의 문자가 1개 이상 존재
{숫자}	숫자만큼 반복
{숫자1, 숫자2}	숫자1 이상 숫자2 이하만큼 반복
{숫자,}	숫자 이상만큼 반복
|	A|B와 같이 쓰이며 A 또는 B의 의미를 가짐
( )	그룹을 만듦. 예: (x)(yz)

패턴 지정 방법 2

패턴	의미
₩₩	역 슬래쉬(\ ) 문자를 의미
₩d	모든 숫자를 의미. [0-9]와 의미가 동일
₩D	숫자를 제외한 모든 문자를 의미. [^0-9]와 의미가 동일
₩s	공백을 의미. [₩t₩n₩r₩f₩v]와 의미가 동일
₩S	공백을 제외한 문자를 의미. [^ ₩t₩n₩r₩f₩v]와 의미가 동일
₩w	문자 또는 숫자를 의미. [a-zA-Z0-9]와 의미가 동일
₩W	문자 또는 숫자가 아닌 문자를 의미. [^a-zA-Z0-9]와 의미가 동일

정규표현식 모듈 함수

모듈 함수	설명
re.compile()	정규표현식의 패턴을 컴파일함. search 할 때 패턴을 지정하지 않게 됨
re.search()	문자열에서 정규 표현식과 정합되는 부분이 있는지 검색
re.match()	문자열의 처음이 정규 표현식과 정합되는지 검색
re.split()	정규표현식을 기준으로 문자열을 분리하여 리스트로 만듦
re.findall()	문자열에서 매치되는 부분들을 찾아서 리스트로 리턴
re.finditer()	문자열에서 매치되는 부분들에 대한 iterator를 리턴
re.sub()	문자열에서 매치되는 부분을 다른 문자열로 대체

compile 함수의 기능

- compile을 사용하지 않는 경우

```
import re
re.search("at", "string data")
```

```
<re.Match object; span=(8, 10), match='at'>
```

- compile을 사용하면 패턴을 저장함

```
import re
r1 = re.compile("at")
r1.search("string data")
```

```
<re.Match object; span=(8, 10), match='at'>
```

정규 표현식 처리 사례

```
import re

text = """100 John    PROF
101 James   STUD
102 Mac   STUD"""

x1=re.split('\s+', text)
x2=re.findall('\d+', text)
x3=re.findall('[A-Z]', text)
x4=re.findall('[A-Z]{4}', text)
x5=re.findall('[A-Z][a-z]+', text)

print(x1)
print(x2)
print(x3)
print(x4)
print(x5)
```

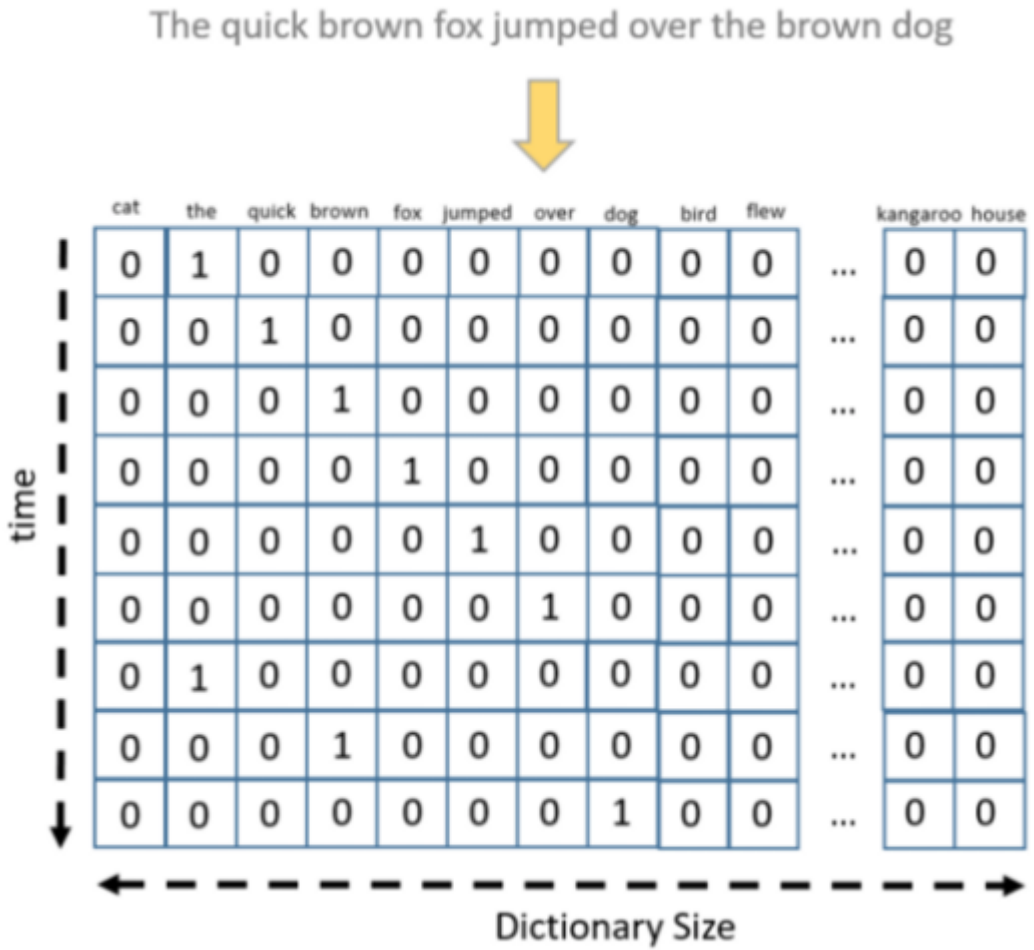
```
['100', 'John', 'PROF', '101', 'James', 'STUD', '102', 'Mac', 'STUD']
['100', '101', '102']
['J', 'P', 'R', 'O', 'F', 'J', 'S', 'T', 'U', 'D', 'M', 'S', 'T', 'U', 'D']
['PROF', 'STUD', 'STUD']
['John', 'James', 'Mac']
```

정수 인코딩

- 기계 번역처럼 컴퓨터로 텍스트를 처리하는 프로그램을 만들 때 각 단어에 고유한 정수를 대응시키는 것이 필요함
- 텍스트에 단어가 10,000개가 있으면 단어 각각에 1~10,000까지의 숫자를 할당하게 됨
- 숫자는 랜덤하게 부여할 수도 있고 빈도수가 높은 순서로 부여하기도 함
- 교재에서는 단어들의 빈도수를 구하는 사례를 보여주고 있음

원-핫 인코딩(One-hot encoding)

- **원-핫 인코딩**은 단어 집합의 크기를 벡터의 차원으로 하고, 단어별로 한 개의 인덱스만 1, 나머지는 0이 되도록 표현하는 방식임



원-핫 인코딩(One-hot encoding)

- 사례: 단어의 수가 10개이고 '나 ' 라는 단어의 고유 숫자가 3이라면 '나 ' 는 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]로 코딩됨
- 단어의 개수가 많으면 벡터의 차원이 커짐
- 이 방법으로는 단어간의 유사성을 표현하지 못함
- 단어의 잠재 의미를 반영하여 다차원 공간에 벡터화하는 방식은 **워드 임베딩**이며, 머신 러닝 기법을 이용하여 벡터 데이터를 구축

한글 인코딩 사례

- 형태소 분석을 통해 문장을 나눈 다음, 발생 순서별로 고유 인덱스를 부여
- one_hot_encoding 함수에서 고유 벡터를 리턴

```
from konlpy.tag import Okt

# 문장을 분석하여 단어들을 token에 저장
okt=Okt()
token=okt.morphs("나는 자연어 처리를 배운다")
print(token)

word2index={}     # word-index를 사전으로 저장
for voca in token:
    if voca not in word2index.keys():
        word2index[voca]=len(word2index)
print(word2index)
```

```
def one_hot_encoding(word, word2index):
        one_hot_vector = [0]*(len(word2index))
        index=word2index[word]
        one_hot_vector[index]=1
        return one_hot_vector

one_hot_encoding("자연어", word2index)
```

2장 참고자료

- **교재 1:** *딥 러닝을 이용한 자연어 처리 입문*, 유원준, 온라인 문서
  - 2장

- *자연어처리 바이블*, 임희석, 2019.
  - 3장. 언어학의 기본 원리

- KoNLPy: 파이썬 한국어 NLP
  - KoNLPy: 파이썬 한국어 NLP — KoNLPy 0.4.3 documentation (konlpy-ko.readthedocs.io)