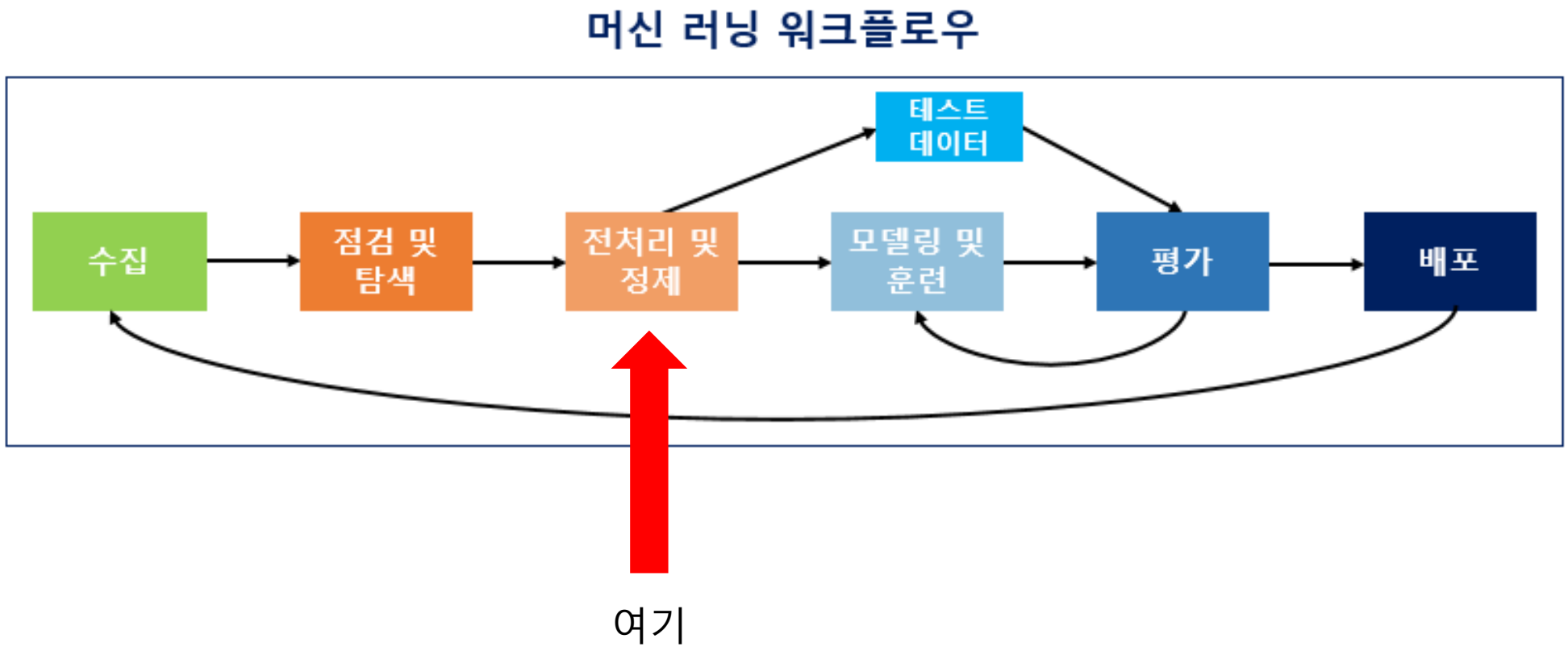
2장 텍스트 전처리

Text preprocessing

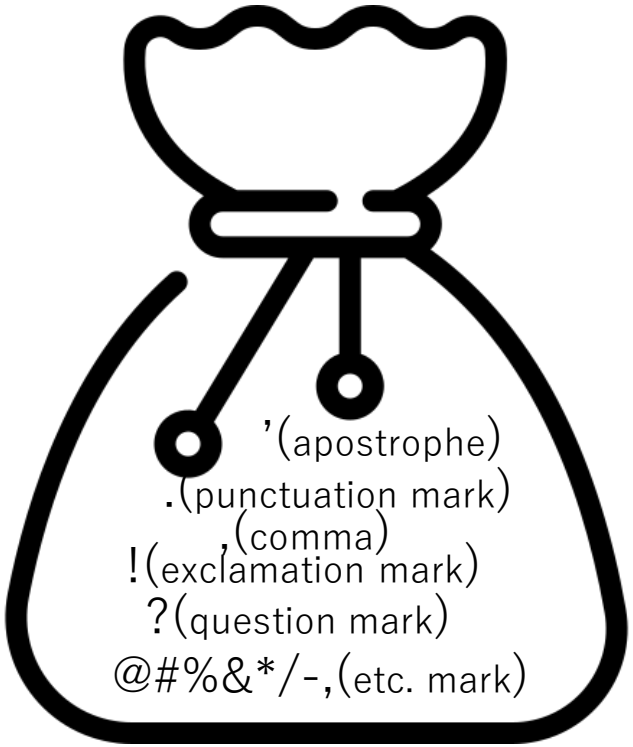
엄호용

목차 Table of Contents

01 토큰화 Tokenization



01 토큰화 Tokenization



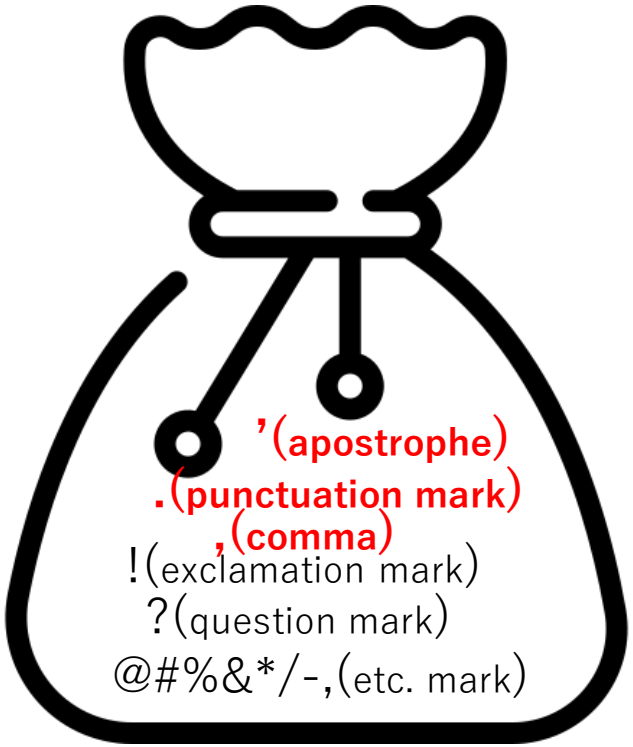
Corpus(말뭉치)

Don’t do that!

Mr. Jone’s dog barked 1,000,000 times.

The train, car and bus does not leave at 12 A.M.

01 토큰화 Tokenization



Corpus(말뭉치)

Don’t do that!

Don’t
Don t
Dont
Do n’t

Jone's
Jone s
Jones
Jone 's

Mr. **Jone’s** dog barked **1,000,000** times.

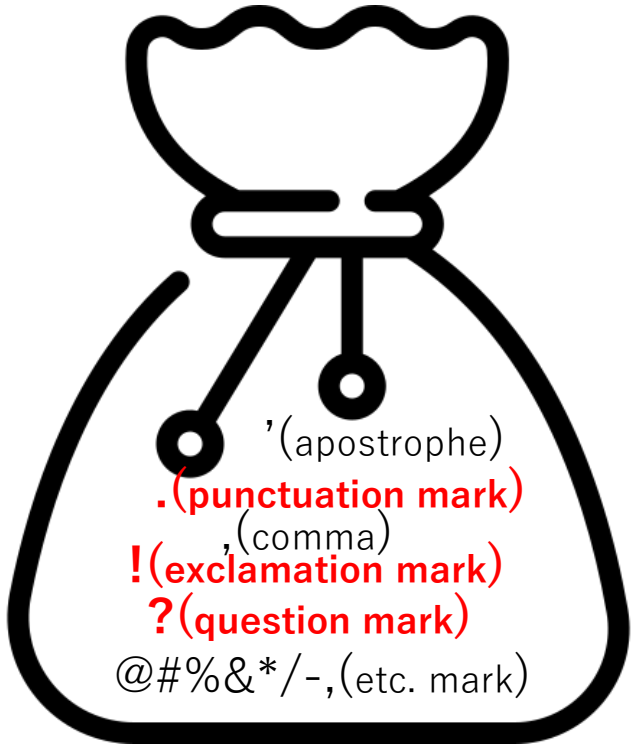
The **train, car and bus** does not leave at 12 **A.M.**

train car and bus
train, car and bus

A.M.
AM

1 000 000
1,000,000

01 토큰화 Tokenization

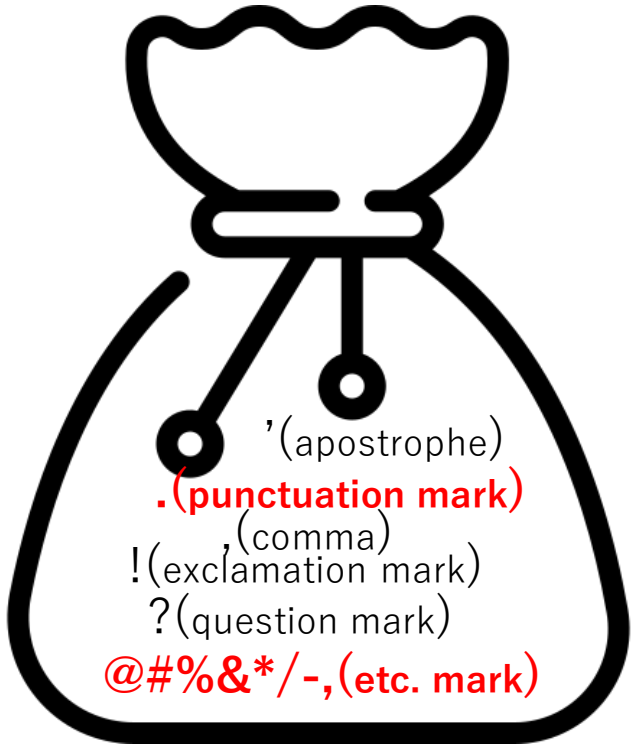


Corpus(말뭉치)

I am happy. Because I like apple, banana, coconut! What about you Dr. Park?

"It was not to them (not to Hugh, or Richard, or even to devoted Miss Brush) the liberator of the pent egotism, which is a strong martial woman, well nourished, well descended, of direct impulses, downright feelings, and little introspective power (broad and simple–why could not every one be broad and simple? she asked) feels rise within her, once youth is past, and must eject upon some object–it may be Emigration, it may be Emancipation; but whatever it be, this object round which the essence of her soul is daily secreted, becomes inevitably prismatic, lustrous, half looking glass, half precious stone; now carefully hidden in case people should sneer at it; now proudly displayed."

01 토큰화 Tokenization



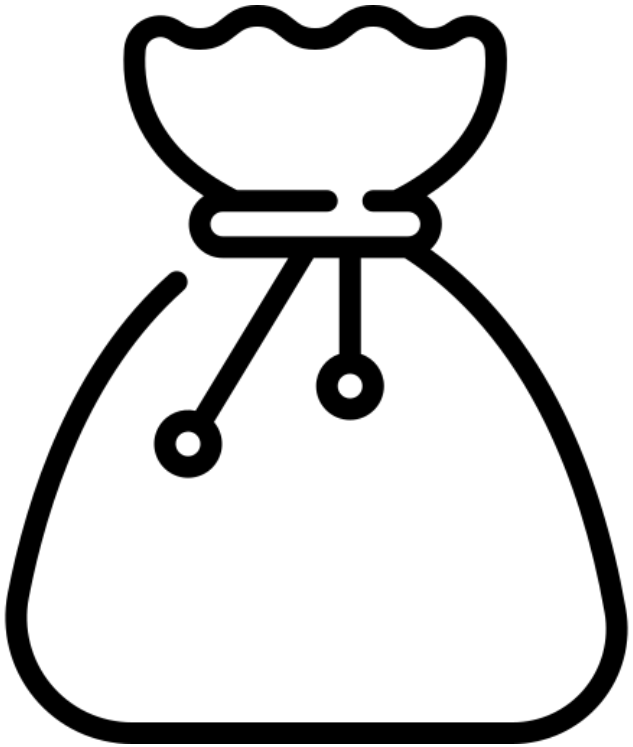
Corpus(말뭉치)

!@#$%^&*()_~.,?/=₩

IP 192.168.56.31 서버에 들어가서 2023/06/08 로그 파일
상위 10%만 잘라내서 aaa@gmail.com로 결과 좀 보내줘.

Starting a home-based restaurant may be an ideal.
It doesn't have a food chain or restaurant of their own.

01 토큰화 Tokenization



Corpus(말뭉치)

그는 그가 그에게 그를 넘겼다는 것을 깨달았다.

→ 인칭대명사 + 조사

‘그’라는 동일한 인칭대명사가 각기 다른 단어로 인식될 수 있다!

한국어는띄어쓰기를안지켜도대충알아먹지롱~

대부분의 경우 띄어쓰기가 잘 지켜지지 않음!

잠자리는 잠자리에 들었다.

동음이의어를 처리해주어야 한다!

01 토큰화 Tokenization

Word Tokenization

도구명	Don’t	Jone’s
word_tokenize	[‘Do’, ”n’t”]	['Jone', "'s"]
WordPunctTokenizer	['Don', "'", 't’]	['Jone', "'", 's’]
text_to_word_sequence	["don’t”]	["jone’s”]
TreebankWordTokenizer	[‘do’, “n’t”]	[‘Jone’, “’s”]

Sentence Tokenization

도구명	대상 언어
sent_tokenize	English
kss	Korean

형태소 분석 및 품사 태깅

도구명	대상
pos_tag	English, 품사
Okt	Korean, 품사 & 형태소
Kkma	Korean, 품사 & 형태소
Mecab	Korean, 품사 & 형태소
Komoran	Korean, 품사 & 형태소
Hannanum	Korean, 품사 & 형태소

02 정제와 정규화 Cleaning and Normalization

- 정규화(normalization) : 표현 방법이 다른 단어들을 통합시켜서 같은 단어로 만들어준다.

    1) 대소문자 통합

    2) 표제어 추출(lemmatization)

    3) 어간 추출(stemming)

- 정제(cleaning) : 갖고 있는 말뭉치로부터 노이즈 데이터를 제거한다.

    1) 불용어 제거

    2) 등장 빈도가 적은 단어 제거

    3) (영어) 길이가 짧은 단어
    제거

03 어간 추출과 표제어 추출 Stemming and Lemmatization

- 표제어(Lemma)?

단어의 기본형

ex) 달리는 → 달리다, has → have, …

- 표제어 추출 WordNetLemmatizer

형태소를 분석하여 어간과 접사를 분리하는 과정을 거친다. (형태학적 파싱)
단어의 품사를 알아야 정확한 결과를 얻을 수 있다.
어간(stem) + 접사(affix)

ex) 달리는 → 달리- + -는, notation → nota- + -tion

03 어간 추출과 표제어 추출 Stemming and Lemmatization

- 어간 추출(Stemming)? PorterStemmer, LancasterStemmer

영어: 어간(stem) + 접사(affix)

ex) notation → **nota**- + -tion

ex) organization → **organ**- + -ization

ex) organ → **organ**

한국어(용언): 어간(stem) + 어미(ending)

ex) 달리는 → 달리- + -는,

ex) 도우면 → 돕/도우- + -면,

- 어간 추출과 표제어 추출의 장단점

* 표제어 추출

문맥이 주어지면 품사 정보가 보존됨.

속도가 비교적 느림.

* 어간 추출

불규칙 용언은 별도의 처리가 필요함.

일반화가 지나치게 되거나 덜 될 수 있음.

품사 정보가 보존되지 않음.

속도가 비교적 빠름.

02 정제와 정규화 Cleaning and Normalization

- 정규화(normalization) : 표현 방법이 다른 단어들을 통합시켜서 같은 단어로 만들어준다.

    1) 대소문자 통합

    2) 표제어 추출(lemmatization)

    3) 어간 추출(stemming)

- 정제(cleaning) : 갖고 있는 말뭉치로부터 노이즈 데이터를 제거한다.

    1) (영어) 길이가 짧은 단어 제거 ex) at, on, to, in, by, a, an

    2) 등장 빈도가 적은 단어 제거

    3) 불용어 제거

04 불용어 Stopword

- NLTK의 불용어 목록 사용(영어)

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from konlpy.tag import Okt
```

```
example = "Family is not an important thing. It's everything."
stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example)

result = []
for word in word_tokens:
    if word not in stop_words:
        result.append(word)

print('불용어 제거 전 :',word_tokens)
print('불용어 제거 후 :',result)
```

```
불용어 제거 전 : ['Family', 'is', 'not', 'an', 'important', 'thing', '.', 'It', "'s", 'everything',
'.']
불용어 제거 후 : ['Family', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
```

- konlpy의 Okt의 불용어 목록 사용(한국어)

```
okt = Okt()

example = "고기를 아무렇게나 구우려고 하면 안 돼. 고기라고 다 같은 게 아니거든. 예컨대 삼겹살을 구울 때는 중요한 게 있지."
stop_words = "를 아무렇게나 구 우려 고 안 돼 같은 게 구울 때 는"

stop_words = set(stop_words.split(' '))
word_tokens = okt.morphs(example)

result = [word for word in word_tokens if not word in stop_words]

print('불용어 제거 전 :',word_tokens)
print('불용어 제거 후 :',result)
```

```
불용어 제거 전 : ['고기', '를', '아무렇게나', '구', '우려', '고', '하면', '안', '돼', '.', '고기', '라고', '다',
'같은', '게', '아니거든', '.', '예컨대', '삼겹살', '을', '구울', '때', '는', '중요한', '게', '있지', '.']
불용어 제거 후 : ['고기', '하면', '.', '고기', '라고', '다', '아니거든', '.', '예컨대', '삼겹살', '을', '중요한',
'있지', '.']
```

05 정규 표현식 Regular Expression

1) 정규 표현식 문법

정규 표현식을 위해 사용되는 문법 중 특수 문자들은 아래와 같습니다.

특수 문자	설명
`.`	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 `\n`는 제외)
`?`	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)
`*`	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)
`+`	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)
`^`	뒤의 문자열로 문자열이 시작됩니다.
`$`	앞의 문자열로 문자열이 끝납니다.
`{숫자}`	숫자만큼 반복합니다.
`{숫자1, 숫자2}`	숫자1 이상 숫자2 이하만큼 반복합니다. `?`, `*`, `+`를 이것으로 대체할 수 있습니다.
`{숫자,}`	숫자 이상만큼 반복합니다.
`[ ]`	대괄호 안의 문자들 중 한 개의 문자와 매치합니다. [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다. `[a-z]`와 같이 범위를 지정할 수도 있습니다. `[a-zA-Z]`는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.
`[^문자]`	해당 문자를 제외한 문자를 매치합니다.
`|`	`A|B`와 같이 쓰이며 A 또는 B의 의미를 가집니다.

문자 규칙	설명
`\\\`	역 슬래쉬 문자 자체를 의미합니다
`\\d`	모든 숫자를 의미합니다. `[0-9]`와 의미가 동일합니다.
`\\D`	숫자를 제외한 모든 문자를 의미합니다. `[^0-9]`와 의미가 동일합니다.
`\\s`	공백을 의미합니다. `[ \t\n\r\f\v]`와 의미가 동일합니다.
`\\S`	공백을 제외한 문자를 의미합니다. `[^ \t\n\r\f\v]`와 의미가 동일합니다.
`\\w`	문자 또는 숫자를 의미합니다. `[a-zA-Z0-9]`와 의미가 동일합니다.
`\\W`	문자 또는 숫자가 아닌 문자를 의미합니다. `[^a-zA-Z0-9]`와 의미가 동일합니다.

05 정규 표현식 Regular Expression

1) 정규 표현식 문법

정규 표현식을 위해 사용되는 문법 중 특수 문자들은 아래와 같습니다.

특수 문자	설명
`.`	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 `\n`는 제외)
`?`	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)
`*`	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)
`+`	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)
`^`	뒤의 문자열로 문자열이 시작됩니다.
`$`	앞의 문자열로 문자열이 끝납니다.
`{숫자}`	숫자만큼 반복합니다.
`{숫자1, 숫자2}`	숫자1 이상 숫자2 이하만큼 반복합니다. `?`, `*`, `+`를 이것으로 대체할 수 있습니다.
`{숫자,}`	숫자 이상만큼 반복합니다.
`[ ]`	대괄호 안의 문자들 중 한 개의 문자와 매치합니다. [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다. `[a-z]`와 같이 범위를 지정할 수도 있습니다. `[a-zA-Z]`는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.
`[^문자]`	해당 문자를 제외한 문자를 매치합니다.
`|`	`A|B`와 같이 쓰이며 A 또는 B의 의미를 가집니다.

```
r = re.compile("a.c")
r.search("kkk") # 아무런 결과도 출력되지 않는다.
```

```
r.search("abc")
```

```
<_sre.SRE_Match object; span=(0, 3), match='abc'>
```

05 정규 표현식 Regular Expression

1) 정규 표현식 문법

정규 표현식을 위해 사용되는 문법 중 특수 문자들은 아래와 같습니다.

특수 문자	설명
`.`	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 `\n`는 제외)
`?`	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)
`*`	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)
`+`	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)
`^`	뒤의 문자열로 문자열이 시작됩니다.
`$`	앞의 문자열로 문자열이 끝납니다.
`{숫자}`	숫자만큼 반복합니다.
`{숫자1, 숫자2}`	숫자1 이상 숫자2 이하만큼 반복합니다. `?`, `*`, `+`를 이것으로 대체할 수 있습니다.
`{숫자,}`	숫자 이상만큼 반복합니다.
`[ ]`	대괄호 안의 문자들 중 한 개의 문자와 매치합니다. [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다. `[a-z]`와 같이 범위를 지정할 수도 있습니다. `[a-zA-Z]`는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.
`[^문자]`	해당 문자를 제외한 문자를 매치합니다.
`|`	`A|B`와 같이 쓰이며 A 또는 B의 의미를 가집니다.

```
r = re.compile("ab?c")
r.search("abbc") # 아무런 결과도 출력되지 않는다.
```

```
r.search("abc")
```

```
<_sre.SRE_Match object; span=(0, 3), match='abc'>
```

```
r.search("ac")
```

```
<_sre.SRE_Match object; span=(0, 2), match='ac'>
```

05 정규 표현식 Regular Expression

1) 정규 표현식 문법

정규 표현식을 위해 사용되는 문법 중 특수 문자들은 아래와 같습니다.

특수 문자	설명
`.`	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 `\n`는 제외)
`?`	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)
`*`	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)
`+`	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)
`^`	뒤의 문자열로 문자열이 시작됩니다.
`$`	앞의 문자열로 문자열이 끝납니다.
`{숫자}`	숫자만큼 반복합니다.
`{숫자1, 숫자2}`	숫자1 이상 숫자2 이하만큼 반복합니다. `?`, `*`, `+`를 이것으로 대체할 수 있습니다.
`{숫자,}`	숫자 이상만큼 반복합니다.
`[ ]`	대괄호 안의 문자들 중 한 개의 문자와 매치합니다. [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다. `[a-z]`와 같이 범위를 지정할 수도 있습니다. `[a-zA-Z]`는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.
`[^문자]`	해당 문자를 제외한 문자를 매치합니다.
`|`	`A|B`와 같이 쓰이며 A 또는 B의 의미를 가집니다.

```
r = re.compile("ab+c")
r.search("ac") # 아무런 결과도 출력되지 않는다.
```

```
r.search("abc")
```

```
<_sre.SRE_Match object; span=(0, 3), match='abc'>
```

```
r.search("abbbbc")
```

```
<_sre.SRE_Match object; span=(0, 6), match='abbbbc'>
```

05 정규 표현식 Regular Expression

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

```
re.findall('\d+',text)
```

```
['100', '101', '102]
```

```
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("[\w]+")
tokenizer2 = RegexpTokenizer("\s+", gaps=True)

print(tokenizer1.tokenize(text))
print(tokenizer2.tokenize(text))
```

```
['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']
```

06 정수 인코딩 Integer Encoding

말뭉치(corpus)

```
raw_text = "A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret! The Secret He Kept is huge secret. Huge secret. His barber kept his word. a barber kept his word. His barber kept his secret. But keeping and keeping such a huge secret to himself was driving the barber crazy. the barber went up a huge mountain."
```

빈도순으로 정렬

```
[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3), ('word', 2), ('keeping', 2), ('good', 1), ('knew', 1), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)]
```

인덱스 부여

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7}
```

정수 인코딩

```
[[1, 5], [1, 6, 5], [1, 3, 5], [6, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [6, 6, 3, 2, 6, 1, 6], [1, 6, 3, 6]]
```

07 패딩 Padding

정수 인코딩

```
[[1, 5], [1, 6, 5], [1, 3, 5], [6, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [6, 6, 3, 2, 6, 1, 6], [1, 6, 3, 6]]
```

제로 패딩(zero padding)

```
array([[ 1,  5,  0,  0,  0,  0,  0],
       [ 1,  8,  5,  0,  0,  0,  0],
       [ 1,  3,  5,  0,  0,  0,  0],
       [ 9,  2,  0,  0,  0,  0,  0],
       [ 2,  4,  3,  2,  0,  0,  0],
       [ 3,  2,  0,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  2,  0,  0,  0,  0],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 1, 12,  3, 13,  0,  0,  0]])
```

단어 개수 + 1로 패딩

```
array([[ 1,  5, 14, 14, 14, 14, 14],
       [ 1,  8,  5, 14, 14, 14, 14],
       [ 1,  3,  5, 14, 14, 14, 14],
       [ 9,  2, 14, 14, 14, 14, 14],
       [ 2,  4,  3,  2, 14, 14, 14],
       [ 3,  2, 14, 14, 14, 14, 14],
       [ 1,  4,  6, 14, 14, 14, 14],
       [ 1,  4,  6, 14, 14, 14, 14],
       [ 1,  4,  2, 14, 14, 14, 14],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 1, 12,  3, 13, 14, 14, 14]], dtype=int32)
```

손실이 있는 패딩

```
array([[ 1,  5,  0,  0,  0],
       [ 1,  8,  5,  0,  0],
       [ 1,  3,  5,  0,  0],
       [ 9,  2,  0,  0,  0],
       [ 2,  4,  3,  2,  0],
       [ 3,  2,  0,  0,  0],
       [ 1,  4,  6,  0,  0],
       [ 1,  4,  6,  0,  0],
       [ 1,  4,  2,  0,  0],
       [ 7,  7,  3,  2, 10],
       [ 1, 12,  3, 13,  0]], dtype=int32)
```

08 원-핫 인코딩 One-Hot Encoding

```
text = "나랑 점심 먹으러 갈래 점심 메뉴는 햄버거 갈래 갈래 햄버거 최고야"
```

단어 집합 (Vocabulary)

: 서로 다른 단어의 집합;
동일한 단어가 다른 형태로
활용된 것도 '서로 다르다'고 간주한다.
ex) {'book', 'books'}

```
단어 집합 : {'갈래': 1, '점심': 2, '햄버거': 3, '나랑': 4, '먹으러': 5,
'메뉴는': 6, '최고야': 7}
```

```
[2, 5, 1, 6, 3, 7]
```

```
[[0. 0. 1. 0. 0. 0. 0. 0.] # 인덱스 2의 원-핫 벡터
 [0. 0. 0. 0. 0. 1. 0. 0.] # 인덱스 5의 원-핫 벡터
 [0. 1. 0. 0. 0. 0. 0. 0.] # 인덱스 1의 원-핫 벡터
 [0. 0. 0. 0. 0. 0. 1. 0.] # 인덱스 6의 원-핫 벡터
 [0. 0. 0. 1. 0. 0. 0. 0.] # 인덱스 3의 원-핫 벡터
 [0. 0. 0. 0. 0. 0. 0. 1.]] # 인덱스 7의 원-핫 벡터
```

09 데이터의 분리 Splitting Data

X : 독립 변수 데이터. (배열이나 데이터프레임)
Y : 종속 변수 데이터. 레이블 데이터.
X는 문제지, Y는 정답지에 비유할 수 있음.

[[X_1, Y_1], [X_2, Y_2], ... , [X_n, Y_n]] 꼴로 나타냄

```
values = [['당신에게 드리는 마지막 혜택!', 1],
['내일 뵐 수 있을지 확인 부탁드...', 0],
['도연씨. 잘 지내시죠? 오랜만입...', 0],
['(광고) AI로 주가를 예측할 수 있다!', 1]]
columns = ['메일 본문', '스팸 메일 유무']

df = pd.DataFrame(values, columns=columns)
df
```

텍스트(메일의 내용)	레이블(스팸 여부)
당신에게 드리는 마지막 혜택! ...	스팸 메일
내일 뵐 수 있을지 확인 부탁...	정상 메일
...	...
(광고) 멋있어질 수 있는...	스팸 메일

10 한국어 전처리 패키지 Text Preprocessing Tools for Korean Text

1. PyKoSpacing(띄어쓰기 안 한 문장을 띄어쓰기 한 문장으로 변환해주는 패키지)
2. Py-Hanspell(네이버 한글 맞춤법 검사기를 바탕으로 만들어진 패키지)
3. SOYNLP를 이용한 단어 토큰화
4. SOYNLP를 이용한 반복되는 문자 정제
5. Customized KoNLPy(커스텀 토큰 추가 가능)

SOYNLP

비지도 학습을 통한 단어 토큰화

응집 확률(**cohesion probabilitv**)

- $cohesion(2) = P(\text{반포}|\text{반})$
- $cohesion(3) = \sqrt[2]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}))}$
- $cohesion(4) = \sqrt[3]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}) \cdot P(\text{반포한강}|\text{반포한}))}$
- $cohesion(5) = \sqrt[4]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}) \cdot P(\text{반포한강}|\text{반포한}) \cdot P(\text{반포한강공}|\text{반포한강}))}$
- $cohesion(6) = \sqrt[5]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}) \cdot P(\text{반포한강}|\text{반포한}) \cdot P(\text{반포한강공}|\text{반포한강}) \cdot P(\text{반포한강공원}|\text{반포한강공}))}$
- $cohesion(7) = \sqrt[5]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}) \ldots \text{중략} \ldots \cdot P(\text{반포한강공원}|\text{반포한강공}) \cdot P(\text{반포한강공원에}|\text{반포한강공원}))}$

$$\text{cohesion}(n) = \left(\prod_{i=1}^{n-1} P(c_{1:i+1}|c_{1:i}) \right)^{\frac{1}{n-1}}$$

```
word_score_table["반포한강공"].cohesion_forward
```

```
0.2972877884078849
```

```
word_score_table["반포한강공원"].cohesion_forward
```

```
0.37891487632839754
```

```
word_score_table["반포한강공원에"].cohesion_forward
```

```
0.33492963377557666
```

브랜칭 엔트로피(**branching entropy**)

```
word_score_table["디스"].right_branching_entropy
```

```
1.6371694761537934
```

```
word_score_table["디스플"].right_branching_entropy
```

```
-0.0
```

```
word_score_table["디스플레이"].right_branching_entropy
```

```
3.1400392861792916
```

SOYNLP

SOYNLP의 L tokenizer

```
from soynlp.tokenizer import LTokenizer

scores = {word:score.cohesion_forward for word, score in word_s
core_table.items()}
l_tokenizer = LTokenizer(scores=scores)
l_tokenizer.tokenize("국제사회와 우리의 노력들로 범죄를 척결하자", flatte
n=False)
```

Copy

```
[('국제사회', '와'), ('우리', '의'), ('노력', '들로'), ('범죄', '를'),
('척결', '하자')]
```

SOYNLP

```
from soynlp.tokenizer import MaxScoreTokenizer

maxscore_tokenizer = MaxScoreTokenizer(scores=scores)
maxscore_tokenizer.tokenize("국제사회와우리의노력들로범죄를척결하자")
```

```
['국제사회', '와', '우리', '의', '노력', '들로', '범죄', '를', '척결',
'하자']
```

```
print(emoticon_normalize('앜ㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠㅠ', num_repeats
=2))
print(emoticon_normalize('앜ㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠ', num_r
epeats=2))
print(emoticon_normalize('앜ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠㅠ
ㅠ', num_repeats=2))
print(emoticon_normalize('앜ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠ
ㅠㅠㅠㅠㅠㅠㅠ', num_repeats=2))
```

감사합니다!