Tokenization

Contents

- 토큰화(Tokenization)
- 단어 토큰화(Word Tokenization)
- 문장 토큰화(Sentence Tokenization)
- 영어에서의 토큰화(Tokenization in English)
- 한국어에서의 토큰화(Tokenization in Korean)

토큰화(Tokenization)

- 토큰화(Tokenization)는 컴퓨터에게 어디까지가 단어이고 어디까지가 문장인지를 알려주는 과정입니다.
- 단어 토큰화(Word Tokenization), 문장 토큰화(Sentence Tokenization), 그리고 하위단어 토큰화(Subword Tokenization)와 같은 다양한 토큰화 방법이 존재합니다.
- 아래는 한국어 문장을 공백을 기준으로 단어 토큰화(Word Tokenization)를 진행한 예시입니다.

```
In [1]: sentence = '아주대학교 일반 대학원 수학과 데이터 사이언스 전공에 소속된 이규철입니다.'
        word_tokenization = sentence.split()
        print(word_tokenization)

        ['아주대학교', '일반', '대학원', '수학과', '데이터', '사이언스', '전공에', '소속된', '이규철입니다.']
```

단어 토큰화(Word Tokenization)

- 토큰의 단위가 단어일 경우 단어 토큰화(Word Tokenization)라고 합니다.
- 아래는 영어 문장을 공백을 기준으로 단어 토큰화(Word Tokenization)를 진행한 예시입니다.

In [2]:
```
sentence = 'I started studying natural language processing.'
word_tokenization = sentence.split()
print(word_tokenization)
```

```
['I', 'started', 'studying', 'natural', 'language', 'processing.']
```

단어 토큰화(Word Tokenization)

- “We are students!”와 “We’re students!!”라는 의미가 동일한 두 문장이 있습니다.
- 의미가 동일한 두 문장에 대해, 공백을 기준으로 단어를 나누면 의미가 동일하더라도 각각 다른 결과를 가져옵니다.
- 예를들어 "students!”와 "students!!"는 의미가 동일하지만 파이썬에서는 다른 단어로 간주됩니다.

```
In [3]: sentence1 = "We are students!"
        sentence2 = "We're students!!"
        word_tokenization1 = sentence1.split()
        word_tokenization2 = sentence2.split()
        print(word_tokenization1)
        print(word_tokenization2)
        print(word_tokenization1[2] == word_tokenization2[1])

        ['We', 'are', 'students!']
        ["We're", 'students!!']
        False
```

단어 토큰화(Word Tokenization)

- "The price of Nintendo is $299.99." 라는 문장이 있을 때, 특수문자가 토큰화(Tokenization)에 방해가 될 수 있으므로 모든 특수문자를 제거하는 규칙을 적용한다면, 원래의 의미가 상실될 수 있습니다.
- 아래는 영어 문장의 특수문자를 제거하고 공백을 기준으로 단어 토큰화(Word Tokenization)를 진행한 예시입니다.

In [4]:
```
import re
sentence = "The price of Nintendo is $299.99."
new_sentence = re.sub(r'[^\w\s]', '', sentence)
word_tokenization = new_sentence.split()
print(word_tokenization)
```

```
['The', 'price', 'of', 'Nintendo', 'is', '29999']
```

단어 토큰화(Word Tokenization)

- 토큰화(Tokenization) 작업은 상당히 섬세하게 규칙을 설계해야 합니다. 단어와 문장의 경계를 정확히 파악하여 제대로된 의미를 유지하고 처리하기 위해 각 언어의 특성과 문맥을 고려해야 합니다. 특히, 특수문자, 줄임말, 구두점 등 다양한 언어의 특징을 고려하여 적절한 토큰화 규칙을 설계해야 합니다.
- 개인이 토큰화(Tokenization) 규칙을 개발하는 대신에 이미 구현된 토크나이저(Tokenizer)를 사용하는 것이 권장됩니다. 이미 구현된 토크나이저(Tokenizer)들은 토큰화(Tokenization) 작업을 매우 간단하고 효율적으로 수행할 수 있도록 정교하게 설계되었습니다.

단어 토큰화(Word Tokenization)

- 영어권 언어의 단어의 토크나이저(Tokenizer)에는 여러가지 선택지가 존재합니다.
- TreebankWordTokenizer는 그 선택지중에 하나입니다.
- TreebankWordTokenizer는 Penn Treebank Tokenization이라는 정해진 규칙을 따릅니다.
- TreebankWordTokenizer의 규칙은 '하이푼(-)'으로 구성된 단어는 하나로 유지하고 doesn't와 같이 '어퍼스트로피(')'로 접어가 함께하는 단어는 분리해줍니다.

단어 토큰화(Word Tokenization)

- nltk는 파이썬에서 자연어 처리 및 문서 분석을 수행하는 데 사용되는 라이브러리입니다.
- 아래는 영어 문장을 TreebankWordTokenizer를 사용해서 단어 토큰화(Word Tokenization)를 진행한 예시입니다.

In [5]:
```
import nltk
from nltk.tokenize import TreebankWordTokenizer

tokenizer = TreebankWordTokenizer()
sentence = "I'm so happy to be with my hamster."
tokens = tokenizer.tokenize(sentence)
print(tokens)
```

```
['I', "'m", 'so', 'happy', 'to', 'be', 'with', 'my', 'hamster', '.']
```

문장 토큰화(Sentence Tokenization)

- 토큰의 단위가 문장일 경우 문장 토큰화(Sentence Tokenization)라고 합니다.
- 문장 토큰화(Sentence Tokenization)의 경우 일반적으로 흔히들 '마침표(.)', '느낌표(!)', '물음표(?)' 등의 구두점으로 문장을 쉽게 구분할 수 있다고 잘못 생각하기도 합니다.
- 실제로 '느낌표(!)', '물음표(?)'는 일반적으로 명확한 구분자의 역할을 해주지만 '마침표(.)'는 그렇지 않습니다.
- 아래는 한국어 텍스트를 '마침표(.)', '느낌표(!)', '물음표(?)'를 기준으로 문장 토큰화(Sentence Tokenization)를 진행한 예시입니다.

```
In [6]: text1 = "아이피 192.168.56.31 서버에 들어가서 로그 파일을 저장해줘. 내 이메일로 결과 좀 보내줘. 밥 먹으러 가자."
        text2 = "햄스터는 너무 귀여워! 너가 더 귀여워! 고마워!"
        text3 = "너 포켓몬스터 게임을 하니? 아니 넌 하니? 너 요즘 어떻게 지내?"
        sentence_tokenization1 = text1.split('.')
        sentence_tokenization2 = text2.split('!')
        sentence_tokenization3 = text3.split('?')
        print(sentence_tokenization1)
        print(sentence_tokenization2)
        print(sentence_tokenization3)

        ['아이피 192', '168', '56', '31 서버에 들어가서 로그 파일을 저장해줘', ' 내 이메일로 결과 좀 보내줘', ' 밥 먹으러 가자', '']
        ['햄스터는 너무 귀여워', ' 너가 더 귀여워', ' 고마워', '']
        ['너 포켓몬스터 게임을 하니', ' 아니 넌 하니', ' 너 요즘 어떻게 지내', '']
```

문장 토큰화(Sentence Tokenization)

- 아래는 한국어 텍스트를 nltk의 sent_tokenize를 사용해서 문장 토큰화(Sentence Tokenization)를 진행한 예시입니다.

```
In [7]: import nltk
        from nltk.tokenize import sent_tokenize

        text1 = "아이피 192.168.56.31 서버에 들어가서 로그 파일을 저장해줘. 내 이메일로 결과 좀 보내줘. 밥 먹으러 가자."
        text2 = "햄스터는 너무 귀여워! 너가 더 귀여워! 고마워!"
        text3 = "너 포켓몬스터 게임을 하니? 아니 넌 하니? 너 요즘 어떻게 지내?"
        sentence_tokenization1 = sent_tokenize(text1)
        sentence_tokenization2 = sent_tokenize(text2)
        sentence_tokenization3 = sent_tokenize(text3)
        print(sentence_tokenization1)
        print(sentence_tokenization2)
        print(sentence_tokenization3)

        ['아이피 192.168.56.31 서버에 들어가서 로그 파일을 저장해줘.', '내 이메일로 결과 좀 보내줘.', '밥 먹으러 가자.']
        ['햄스터는 너무 귀여워!', '너가 더 귀여워!', '고마워!']
        ['너 포켓몬스터 게임을 하니?', '아니 넌 하니?', '너 요즘 어떻게 지내?']
```

- nltk의 sent_tokenize는 기본적으로 영어 문장 토큰화에 최적화되어 있기 때문에, 한국어의 특성을 완전히 반영하지 못할 수 있습니다.

문장 토큰화(Sentence Tokenization)

- 단어 토큰화(Word Tokenization)에서 TreebankWordTokenizer를 사용했던 것처럼 문장 토큰화(Sentence Tokenization)에서도 이미 잘 구현된 토크나이저(Tokenizer)를 사용하는 것이 유리합니다.
- nltk의 sent_tokenize를 사용했을때 “192.168.56.31”에서 '마침표(.)'가 여러 차례 등장함에도 불구하고 잘 구분하여 문장 토큰화(Sentence Tokenization)을 수행하는 것을 확인할 수 있습니다.

영어에서의 토큰화(Tokenization in English)

- 영어는 대소문자를 구분하므로 토큰화(Tokenization)할 때 대소문자를 어떻게 다룰지 결정해야 합니다. 일반적으로는 소문자로 변환하는 것이 좋습니다.

```
In [8]: print('Hello' == 'hello')
```

```
False
```

- 아래는 영어 문장의 대문자를 소문자로 바꿔 TreebankTokenizer를 사용해서 단어 토큰화(Word Tokenization)를 진행한 예시입니다.

```
In [9]: import nltk
        from nltk.tokenize import TreebankWordTokenizer

        sentence = "Hello, World!"
        lower_sentence = sentence.lower()
        print(lower_sentence)

        tokenizer = TreebankWordTokenizer()
        tokens = tokenizer.tokenize(lower_sentence)
        print(tokens)
```

```
hello, world!
['hello', ',', 'world', '!']
```

한국어에서의 토큰화(Tokenization in Korean)

- 영어에서는 띄어쓰기가 잘 지켜지지만 한국어에서는 띄어쓰기가 잘 지켜지지 않는 경우가 많습니다. 따라서 토큰화할 때 띄어쓰기를 정확하게 처리해야 합니다.

In [10]:
```
en_sentence = "I am a graduate student."
new_en_sentence = en_sentence.replace(" ", "")
ko_sentence = "나는 대학원생이다."
new_ko_sentence = ko_sentence.replace(" ", "")
print(new_en_sentence)
print(new_ko_sentence)
```

```
Iamagraduatestudent.
나는대학원생이다.
```

한국어에서의 토큰화(Tokenization in Korean)

- 한국어는 교착어이며, 실질적인 의미를 가지는 어간에 조사나 어미와 같은 문법 형태소들이 결합하여 문법적인 기능이 부여되는 언어입니다. 한국어에는 영어에는 없는 '은', '는', '이', '가', '를' 등과 같은 조사가 존재합니다.
- 아래는 한국어 문장을 단어 토큰화(Word Tokenization)를 진행한 예시입니다.

```
In [11]: text = "나는 햄스터가 좋아. 햄스터는 해바라기씨를 좋아해. 햄스터를 다시 키우고싶어."
         word_tokenization = text.split()
         print(word_tokenization)
         print(word_tokenization[1] == word_tokenization[3])

         ['나는', '햄스터가', '좋아.', '햄스터는', '해바라기씨를', '좋아해.', '햄스터를', '다시', '키우고싶어.']
         False
```

- "햄스터가", "햄스터는", "햄스터를" 등과 같이 동일한 단어를 동일한 형태로 인식하지 못하는 경우가 있습니다.

한국어에서의 토큰화(Tokenization in Korean)

- 교착어인 한국어의 특성으로 인해 한국어는 토크나이저(Tokenizer)로 형태소 분석기를 사용하는 것이 보편적입니다.
- konlpy는 한국어 자연어 처리를 위한 파이썬 라이브러리입니다.
- 아래는 한국어 문장을 konlpy의 형태소 분석기 중 Hannanum을 사용해 명사 추출과 형태소 추출 그리고 품사 부착을 진행한 예시입니다.

In [12]:
```
from konlpy.tag import Hannanum

hannanum = Hannanum()
text = "나는 햄스터가 좋아. 햄스터는 해바라기씨를 좋아해. 햄스터를 다시 키우고싶어."
nouns_tokens = hannanum.nouns(text)
morphs_tokens = hannanum.morphs(text)
pos_tokens = hannanum.pos(text)
print('명사 추출:', nouns_tokens)
print('형태소 추출:', morphs_tokens)
print('품사 부착:', pos_tokens)
```

```
명사 추출: ['나', '햄스터', '햄스터', '해바라기씨', '햄스터']
형태소 추출: ['나', '는', '햄스터', '가', '좋', '아', '.', '햄스터', '는', '해바라기씨', '를', '좋', '아', '하', '어', '.', '햄
스터', '를', '다시', '키우', '고', '싶', '어', '.']
품사 부착: [('나', 'N'), ('는', 'J'), ('햄스터', 'N'), ('가', 'J'), ('좋', 'P'), ('아', 'E'), ('.', 'S'), ('햄스터',
'N'), ('는', 'J'), ('해바라기씨', 'N'), ('를', 'J'), ('좋', 'P'), ('아', 'E'), ('하', 'P'), ('어', 'E'), ('.', 'S'),
('햄스터', 'N'), ('를', 'J'), ('다시', 'M'), ('키우', 'P'), ('고', 'E'), ('싶', 'P'), ('어', 'E'), ('.', 'S')]
```

한국어에서의 토큰화(Tokenization in Korean)

- 아래는 한국어 문장을 konlpy의 형태소 분석기 중 Kkma를 사용해 명사 추출과 형태소 추출 그리고 품사 부착을 진행한 예시입니다.

In [13]:
```
from konlpy.tag import Kkma

kkma = Kkma()
text = "나는 햄스터가 좋아. 햄스터는 해바라기씨를 좋아해. 햄스터를 다시 키우고싶어."
nouns_tokens = kkma.nouns(text)
morphs_tokens = kkma.morphs(text)
pos_tokens = kkma.pos(text)
print('명사 추출:', nouns_tokens)
print('형태소 추출:', morphs_tokens)
print('품사 부착:', pos_tokens)
```

```
명사 추출: ['나', '햄스터', '해바라기', '해바라기씨', '씨']
형태소 추출: ['나', '는', '햄스터', '가', '좋', '아', '.', '햄스터', '는', '해바라기', '씨', '를', '좋아하', '어', '.', '햄스
터', '를', '다시', '키우', '고', '싶', '어', '.']
품사 부착: [('나', 'NP'), ('는', 'JX'), ('햄스터', 'NNG'), ('가', 'JKS'), ('좋', 'VA'), ('아', 'ECD'), ('.', 'SF'), ('햄
스터', 'NNG'), ('는', 'JX'), ('해바라기', 'NNG'), ('씨', 'NNB'), ('를', 'JKO'), ('좋아하', 'VV'), ('어', 'ECS'), ('.',
'SF'), ('햄스터', 'NNG'), ('를', 'JKO'), ('다시', 'MAG'), ('키우', 'VV'), ('고', 'ECE'), ('싶', 'VXA'), ('어', 'ECD'),
('.', 'SF')]
```

한국어에서의 토큰화(Tokenization in Korean)

- 아래는 한국어 문장을 konlpy의 형태소 분석기 중 Komoran을 사용해 명사 추출과 형태소 추출 그리고 품사 부착을 진행한 예시입니다.

In [14]:
```
from konlpy.tag import Komoran

komoran = Komoran()
text = "나는 햄스터가 좋아. 햄스터는 해바라기씨를 좋아해. 햄스터를 다시 키우고싶어."
nouns_tokens = komoran.nouns(text)
morphs_tokens = komoran.morphs(text)
pos_tokens = komoran.pos(text)
print('명사 추출:', nouns_tokens)
print('형태소 추출:', morphs_tokens)
print('품사 부착:', pos_tokens)
```

```
명사 추출: ['햄스터', '햄스터', '해바라기', '씨', '햄스터']
형태소 추출: ['나', '는', '햄스터', '가', '좋', '아', '.', '햄스터', '는', '해바라기', '씨', '를', '좋아하', '아', '.', '햄스
터', '를', '다시', '키우', '고', '싶', '어', '.']
품사 부착: [('나', 'NP'), ('는', 'JX'), ('햄스터', 'NNP'), ('가', 'JKS'), ('좋', 'VA'), ('아', 'EF'), ('.', 'SF'), ('햄스
터', 'NNP'), ('는', 'JX'), ('해바라기', 'NNP'), ('씨', 'NNB'), ('를', 'JKO'), ('좋아하', 'VV'), ('아', 'EF'), ('.', 'S
F'), ('햄스터', 'NNP'), ('를', 'JKO'), ('다시', 'MAG'), ('키우', 'VV'), ('고', 'EC'), ('싶', 'VX'), ('어', 'EF'), ('.',
'SF')]
```

한국어에서의 토큰화(Tokenization in Korean)

- 아래는 한국어 문장을 konlpy의 형태소 분석기 중 Okt를 사용해 명사 추출과 형태소 추출 그리고 품사 부착을 진행한 예시입니다.

In [15]:
```
from konlpy.tag import Okt

okt = Okt()
text = "나는 햄스터가 좋아. 햄스터는 해바라기씨를 좋아해. 햄스터를 다시 키우고싶어."
nouns_tokens = okt.nouns(text)
morphs_tokens = okt.morphs(text)
pos_tokens = okt.pos(text)
print('명사 추출:', nouns_tokens)
print('형태소 추출:', morphs_tokens)
print('품사 부착:', pos_tokens)
```

```
명사 추출: ['나', '햄스터', '햄스터', '해바라기씨', '햄스터', '다시']
형태소 추출: ['나', '는', '햄스터', '가', '좋아', '.', '햄스터', '는', '해바라기씨', '를', '좋아해', '.', '햄스터', '를', '다시',
'키우고싶어', '.']
품사 부착: [('나', 'Noun'), ('는', 'Josa'), ('햄스터', 'Noun'), ('가', 'Josa'), ('좋아', 'Adjective'), ('.', 'Punctuatio
n'), ('햄스터', 'Noun'), ('는', 'Josa'), ('해바라기씨', 'Noun'), ('를', 'Josa'), ('좋아해', 'Adjective'), ('.', 'Punctua
tion'), ('햄스터', 'Noun'), ('를', 'Josa'), ('다시', 'Noun'), ('키우고싶어', 'Verb'), ('.', 'Punctuation')]
```

한국어에서의 토큰화(Tokenization in Korean)

- 아래는 한국어 텍스트를 한국어 문장 토크나이저(Sentence Tokenizer) kss를 사용해서 문장 토큰화(Sentence Tokenization)를 진행한 예시입니다.

In [16]:
```
import kss

text = '햄스터는 너무 귀엽습니다. 수명이 짧은게 단점입니다. 사랑으로 키워줍시다.'
print(kss.split_sentences(text))
```

```
[Kss]: Oh! You have mecab in your environment. Kss will take this as a backend! :D

['햄스터는 너무 귀엽습니다.', '수명이 짧은게 단점입니다.', '사랑으로 키워줍시다.']
```