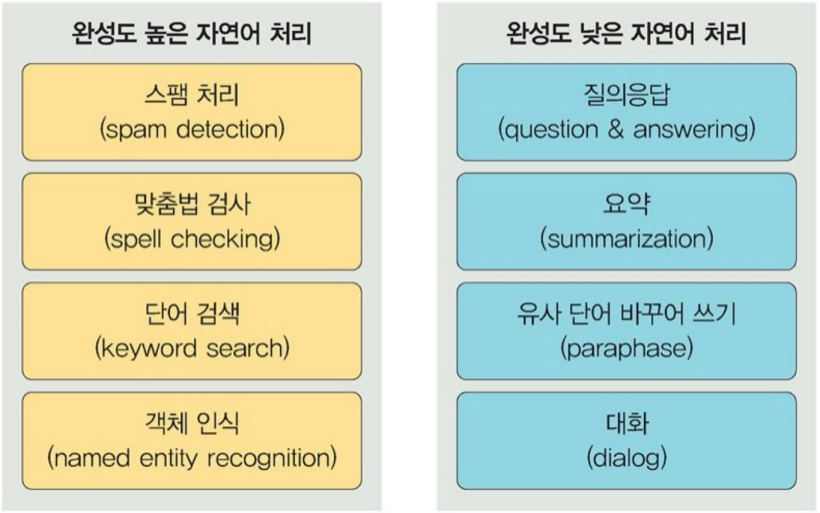
💛

9장) 자연어 전처리

1. 자연어 처리란

1-0. Intro

- 일상생활에서 사용하는 언어 의미를 분석하여 컴퓨터가 처리할 수 있도록 하는 과정
- 자연어 처리가 어려운 이유
  - 딥러닝에 대한 이해 + 인간 언어에 대한 지식 필요
  - 언어 종류가 다르고 그 형태가 다양함 → 임베딩에 어려움
- 자연어 처리가 가능한 영역과 발전 가능한 분야

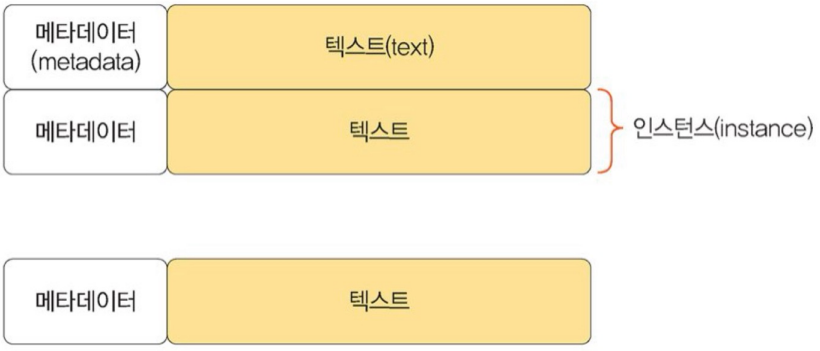


1-1. 자연어 처리 용어 및 과정

관련 용어

말뭉치(corpus)

- 자연어 처리에서 모델을 학습시키기 위한 데이터
- 자연어 연구를 위해 특정한 목적에서 표본을 추출한 집합



토큰(token)

- 자연어 처리를 위해서는 문서를 작은 단위로 나누는 작업이 선행되어야 함
  - 문서를 나누는 단위가 '토큰'
  - 문자열을 토큰으로 분리하는 작업이 '토큰 생성'
  - 문자열을 토큰으로 분리하는 함수를 '토큰 생성 함수'라고 함

토큰화(tokenization)

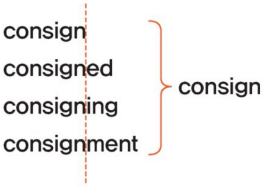
- 텍스트를 문장이나 단어로 분리하는 작업
- 토큰화 단계를 마치면 텍스트가 단어 단위로 분리됨

불용어(stop words)

- 문장 내에서 많이 등장하는 단어 중 분석과 관계없는 단어
- 자주 등장하기에(높은 빈도수) 분석 성능에 영향을 미칠 수 있음
  - 분석 전에 제거해 주어야 함
  - 불용어 예로 "a", "the", "she", "he" 등이 있음

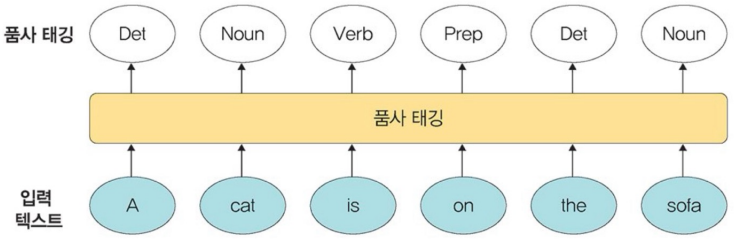
어간 추출(stemming)

- 단어를 기본 형태로 만드는 작업
  - 단어를 기본형으로 통일시키는 작업 등



품사 태깅(part-of-speech tagging)

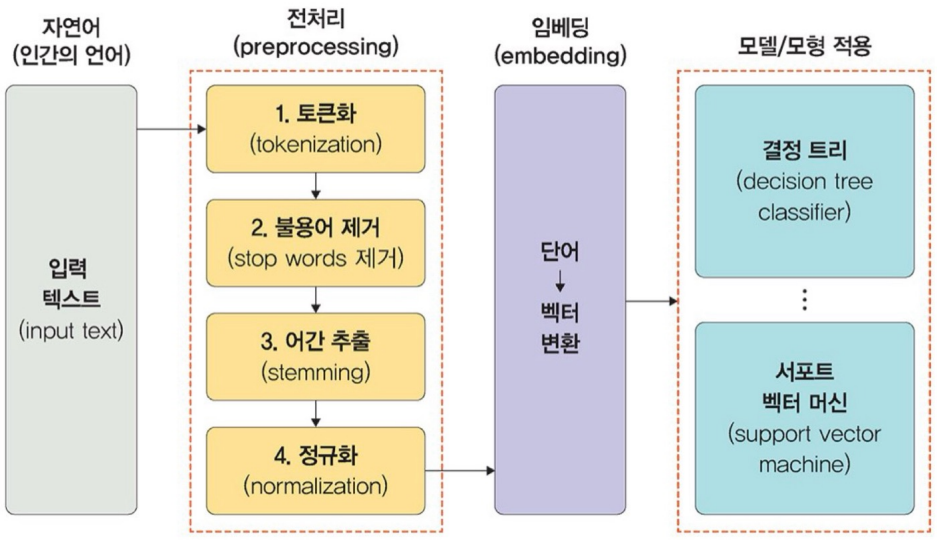
- 주어진 문장에서 품사를 식별하기 위해 붙여 주는 태그(식별 정보)를 의미



- 품사 태깅을 위한 정보
  - Det: 한정사
  - Noun: 명사
  - Verb: 동사
  - Prep: 전치사
- 품사 태그
  - VBZ: 동사, 동명사 또는 현재 분사
  - PRP: 인칭 대명사
  - JJ: 형용사
  - VBG: 동사, 동명사 또는 현재 분사
  - NNS:명사, 복수형
  - CC: 등위 접속사

자연어 처리 과정

- 자연어는 인간의 언어임
  - 인간 언어는 컴퓨터가 이해할 수 없기에 컴퓨터가 이해할 수 있는 언어로 바꿔줘야 함
  - 아래의 4가지 단계를 거침



▲ 그림 9-6 자연어 처리 과정

1. 인간 언어인 자연어를 입력으로 받음
    - 언어마다 처리 방법은 조금씩 다름
    - 현재는 영어에 대한 처리 방법들이 잘 알려져 있음
2. 입력된 텍스트에 대한 전처리 진행
3. 전처리가 완료된 단어들을 임베딩
4. 컴퓨터가 이해할 수 있는 데이터 완성

    → 모델/모형을 이용하여 데이터에 대한 분류, 예측 수행

1-2. 자연어 처리를 위한 라이브러리

NLTK

- 자연어 처리 및 문서 분석용 파이썬 라이브러리
- 주요 기능
    - 말뭉치
    - 토큰 생성
    - 형태소 분석
    - 품사 태깅

KoNLPy

- 한국어 처리를 위한 파이썬 라이브러리

- 파이썬에서 활용 가능한 오픈 소스 형태소 분석기
  - 기존에 공개된 여러 분석기를 한 번에 설치 + 동일한 방법으로 사용 가능하도록 해줌
- 주요 기능
  - 형태소 분석: `morphs()`
  - 품사 태깅: `pos()`

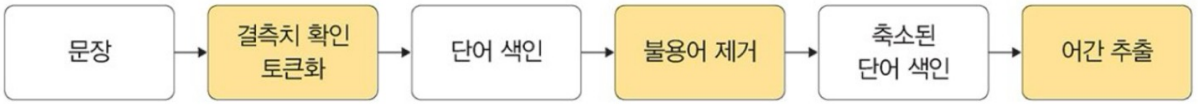
Gensim

- 파이썬에서 제공하는 `Word2Vec` 라이브러리
  - 딥러닝 라이브러리는 아님
  - 효율성 + 확장 가능성으로 인해 폭넓게 사용되고 있음
- 주요 기능
  - 임베딩: Word2Vec
  - 토픽 모델링
  - LDA(Latent Dirichlet Allocation)

사이킷런

- 문서 전처리를 위한 라이브러리를 제공함
  - 특히 특성 추출 용도로 많이 활용
- 주요 기능
  - CountVectorizer: 텍스트에서 단어의 등장 횟수를 기준으로 특성 추출
  - Tfidfvectorizer: TF-IDF 값을 사용하여 텍스트에서 특성 추출
  - HashingVectorizer
    - CountVectorizer와 동일한 방법 활용
    - 텍스트 처리 시 해시 함수 사용 → 더 빠른 처리 속도

2. 전처리



▲ 그림 9-15 전처리 과정

2-1. 결측치 확인

- 값이 없는 데이터를 의미(NaN 등)

결측치 확인하기

- 데이터 직접 출력 후 확인
- `isnull()` 함수 활용

결측치 처리하기

- 모든 행이 결측치인 경우 해당 행을 삭제: `dropna()`
- 결측치를 다른 값으로 채우기: `fillna('채울 데이터')`
- 데이터에 하나라도 NaN 값이 있다면 행 전체를 삭제
- 데이터가 거의 없는 특성의 경우 특성을 삭제(열 삭제)
- 최빈값 또는 평균값 등의 통계치로 대체하기

2-2. 토큰화

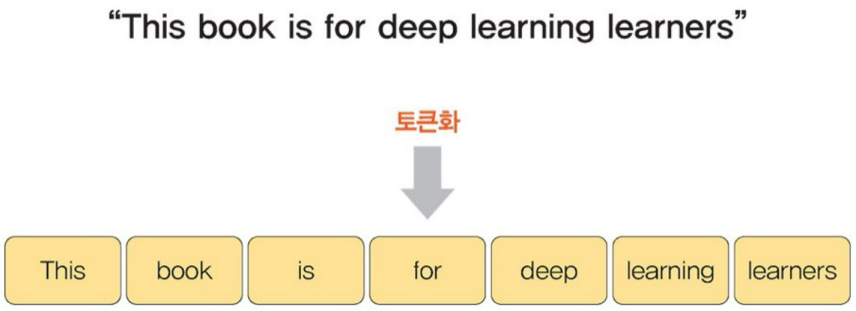
- 주어진 텍스트를 단어/문자 단위로 자르는 것
- 토큰화는 다시 문장 토큰화와 단어 토큰화로 구분됨
- `NLTK` 라이브러리 활용

문장 토큰화

- 마침표, 느낌표, 물음표 등 문장의 마지막을 뜻하는 기호에 따라 문장을 분리하는 것
- `sent_tokenize()`

단어 토큰화

- 띄어쓰기를 기준으로 문장을 구분하는 것



▲ 그림 9-17 단어 토큰화

- 그러나 한국어는 띄어쓰기만으로 토큰을 구분하기 어렵다는 단점이 존재

2-3. 불용어 제거

- 불용어(stop word)
  - 문장 내에서 빈번하게 발생하여 의미를 부여하기 어려운 단어
  - 'a', 'the'와 같은 단어들
- 불용어는 자연어 처리에 있어 효율성을 감소시키고 처리 시간이 길어지는 단점이 있기에 제거가 필요함

2-4. 어간 추출(stemming)

- 단어의 원형을 찾는 작업
  - ex> writing, writes, wrote → write
  - 단어 그 자체만 고려 ⇒ 품사가 달라도 ok
- 표제어 추출(lemmatization)
  - 단어의 원형을 찾는 작업이기는 함
  - 그러나 단어가 문장 속에서 어떤 품사로 쓰였는지도 고려 → 품사가 동일해야 함

⇒ 어간 추출과 표제어 추출 모두 어근 추출이 목적임

⇒ 어간 추출은 사전에 없는 단어도 추출할 수 있는 반면에 표제어 추출은 사전에 있는 단어만 추출 가능

- `nltk.porter`, `nltk.lancaster` 활용
  - 포터 알고리즘의 경우 단어 원형을 비교적 잘 보존함
  - 랭커스터 알고리즘의 경우 단어 원형을 알아볼 수 없을 정도로 축소시킴
    - 낮은 정확도

        - 일반적인 상황보다는 데이터셋을 축소시켜야 하는 특정 상황에서 유용함

표제어 추출

- 표제어 추출은 품사와 같은 문법뿐만 아니라 문장 내에서 단어의 의미 또한 고려
    - 좋은 성능
    - 어간 추출보다 시간이 더 오래 걸리는 단점이 존재
- `nltk.stem.WordNetLemmatizer` 활용
    - 일반적으로 표제어 추출의 성능을 높이기 위해 단어에 대한 품사 정보를 추가하곤 함