Week12_텍스트분석_8장_#1

NLP vs 텍스트분석

✅ **NLP (National Language Processing)**

머신이 인간의 언어를 이해하고 해석하는 데 더 중점을 둠
텍스트 분석을 향상하게 하는 기반 기술

ex) 기계 번역, 자동 질의응답 시스템

텍스트 분석 (텍스트 마이닝)

모델 수립 → 정보 추출 → 분석 작업

- 텍스트 분류 (Text Classification, Text Categorization)

  : 문서가 특정 분류 또는 카테고리에 속하는 것을 예측하는 기법 - 지도학습

- 감성 분석 (Sentiment Analysis)

  : 텍스트에서 나타나는 감정/판단/믿음/의견/기분 등 주관적 요소를 분석하는 기법 - 지도 & 비지도학습

- 텍스트 요약 (Summarization)

  : 텍스트 내에서 중요한 주제나 중심 사상을 추출하는 기법
  ex) 토픽 모델링

- 텍스트 군집화 (Clustering) & 유사도 측정

  : 비슷한 유형의 문서에 대해 군집화를 수행하는 기법

텍스트 분석

= **비정형 데이터**인 텍스트를 분석하는 것

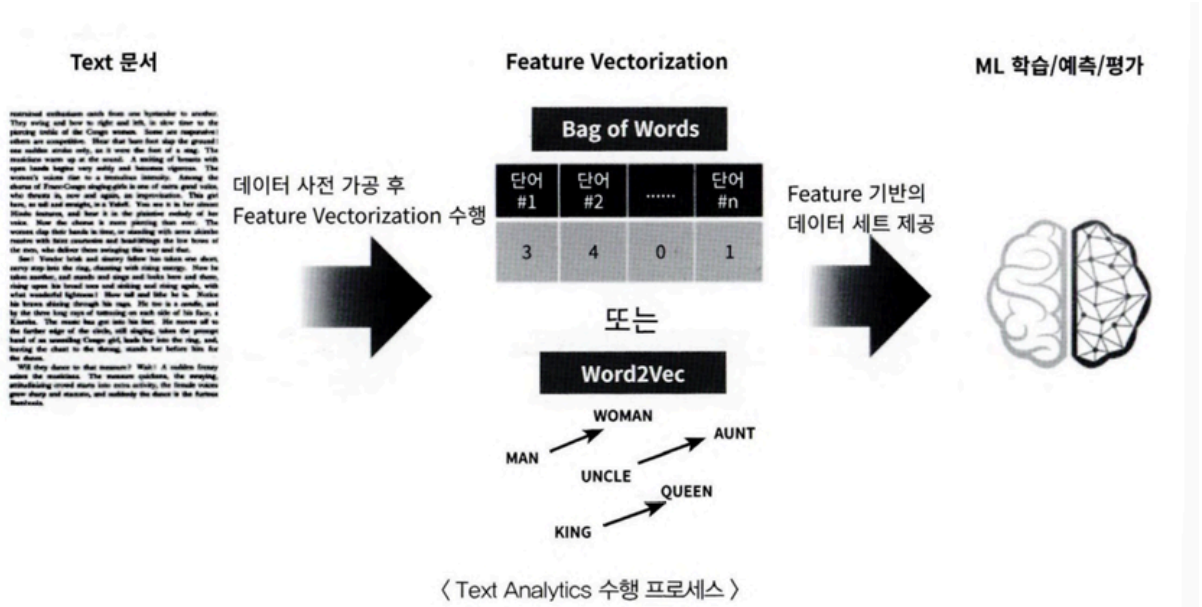
→ 비정형 텍스트 데이터를 피처 형태로 추출하고, 추출된 피처에 의미 있는 값을 부여해야 함

피처 벡터화(Feature Vectorization), 피처 추출(Feature Extraction)

텍스트 → word(또는 word 일부분) 기반의 다수의 피처로 추출 → 피처에 단어 빈도수와 같은 숫자 값 부여
⇒ 텍스트가
단어의 조합인 벡터 값으로 표현됨

ex) BOW(Bag of Words), Word2Vec 방법

텍스트 분석 수행 프로세스



〈 Text Analytics 수행 프로세스 〉

1. **텍스트 전처리**
   : 텍스트를 피처로 만들기 전에 수행하는 작업들

   — 클렌징(대/소문자 변경, 특수문자 삭제 등), 단어 등 토큰화, 의미 없는 단어(Stop word) 제거, 어근 추출(Stemming/Lemmatization) 등 텍스트 정규화 작업
2. **피처 벡터화/추출**
   : 사전 준비 작업으로 가공된 텍스트에서 피처를 추출하고 벡터 값 할당

   — BOW(Count 기반, TF-IDF 기반 벡터화), Word2Vec
3. **ML 모델 수립 및 학습/예측/평가**
   : 피처 벡터화된 데이터 세트에 ML 모델을 적용해 학습/예측 및 평가 수행

파이썬 기반 NLP, 텍스트 분석 패키지

- **NLTK**(Natural Language Toolkit for Python): 가장 대표적인 NLP 패키지. 방대한 데이터 세트 & 서브 모듈. NLP 거의 모든 영역 커버. 수행 속도 측면에서 아쉬움
- Gensim: 토픽 모델링 분야 good
- SpaCy: 뛰어난 수행 성능

+) 결합

텍스트 전처리 - 텍스트 정규화

클렌징(Cleansing)

텍스트에서 분석에 오히려 방해가 되는 불필요한 문자, 기호 등을 사전에 제거하는 작업

ex) HTML, XML 태그, 특정 기호 등

텍스트 토큰화

문장 토큰화 (Sentence Tokenization) - sent_tokenize

마침표, 개행문자 등 문장의 마지막을 뜻하는 기호 또는 정규 표현식에 따른 문장 토큰화

단어 토큰화 (Word Tokenization) - work_tokenize

문장 → 단어

공백, 콤마, 마침표, 개행문자 등으로 단어 분리 또는 정규 표현식 이용

n-gram

: 문장을 단어별로 하나씩 토큰화할 경우 문맥적인 의미가 무시됨

⇒ 연속된 n개의 단어를 하나의 토큰화 단위로 분리해 내자

ex) “Agent Smith knocks the door” → 2-gram(bigram)

⇒ (Agent, Smith), (Smith, knocks), ...

스톱 워드 제거

분석에 큰 의미가 없는 단어 제거

NLTK가 다양한 언어의 스톱 워드 제공해 줌 -
nltk.corpus.stopwords.words('language')

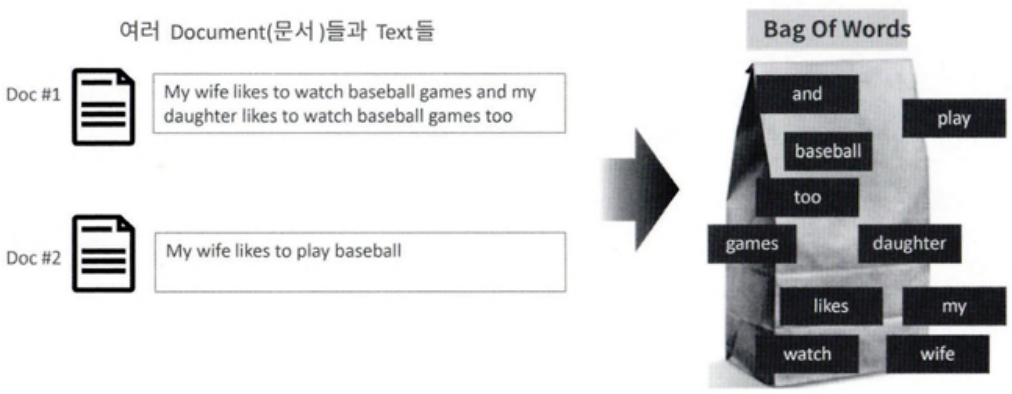
Stemming & Lemmatization

문법적 또는 의미적으로 변화하는 단어의 원형 찾기

- Stemming: 변환 시 일반적인 방법 또는 더 단순화된 방법 적용
  → 원래 단어에서 일부 철자가 훼손된 어근 단어를 추출하는 경향 O
  - NLTK → Porter, Lancaster, Snowball Stemmer
- Lemmatization: 문법적인 요소 & 더 의미적인 부분 감안 → 정확한 철자로 된 어근 단어 찾아줌, 변환 시간 ⬆️
  - NLTK → WordNetLemmatizer

Bag of Words - BOW

문서가 가지는 모든 단어를 문맥 순서 무시하고 일괄적으로 단어에 대해 **빈도 값**을 부여해 피처 값을 추출하는 모델



〈Bag of Words 모델〉

✏️ 예를 들어

문장 1:

'My wife likes to watch baseball games and my daughter likes to watch baseball games too'

문장 2:

'My wife likes to play baseball'

두 문장에서 BOW의 단어 수(Word Count) 기반으로 피처를 추출하면

1. 두 문장의 모든 단어에서 중복 제거하고 각 단어(feature, term)를 칼럼 형태로 나열
   각 단어에 고유 인덱스를 부여함
2. 개별 문장에서 해당 단어가 나타나는 횟수(Occurrence)를 각 단어에 기재

	Index 0	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6	Index 7	Index 8	Index 9	Index 10
	and	baseball	daughter	games	likes	my	play	to	too	watch	wife
문장 1	1	2	1	2	2	2		2	1	2	1
문장 2		1			1	1	1	1			1

문장 1에서 baseball은 2회 나타남

장점

- 쉽고 빠른 구축
- 생각보다 문서 특징을 잘 나타낼 수 있는 모델

단점

- 문맥 의미(Semantic Context) 반영 부족

  : BOW는 단어 순서를 고려하지 X → 문장 내에서 단어의 문맥적인 의미가 무시됨. n-gram 기법을 활용해도 제한이 있음

- 희소 행렬 문제

  : BOW로 피처 벡터화 수행하면 희소 행렬 형태의 데이터가 만들어지기 쉬움. 희소 행렬은 일반적으로 ML 알고리즘의 수행 시간과 예측 성능을 떨어뜨리기 때문에 희소 행렬을 위한 특별한 기법이 마련되어 있음

BOW 피처 벡터화

✅ **피처 벡터화**

: 텍스트를 특정 의미를 가지는 숫자형 값인 벡터 값으로 변환

ex) 각 문서의 텍스트를 단어로 추출해 피처로 할당하고, 각 단어의 발생 빈도와 같은 값을 피처에 값으로 부여

→ 각
문서를 **단어 피처의 발생 빈도 값**으로 구성된 벡터로 만드는 기법

** 기존 텍스트 데이터를 또다른 형태의 피처 조합으로 변경하는 것이므로, 넓은 의미의 피처 추출에 포함됨

✅ BOW 모델에서 피처 벡터화

= 모든 문서에서 모든 단어를 칼럼 형태로 나열
→ 각 문서에서 해당 단어의 횟수나 정규화된 빈도를 값으로 부여하는 데이터 세트 모델로 변경

〈 BOW 피처 벡터화 〉

방식 1 - 카운트 기반 벡터화

단어 피처에 해당 단어가 각 문서에서 나타나는 횟수, Count를 부여

카운트 값 ⬆️ → 인식되는 중요도 ⬆️
⇒ 카운트만 부여하면 스탑 워드같은 것들까지 높은 값이 부여됨

방식 2 - TF-IDF 기반 벡터화

(Term Frequency - Inverse Document Frequency)

개별 문서에서 자주 나타나는 단어에 높은 가중치를 두되, 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 패널티를 주는 방식

한 개의 문서(Document)

모든 문서들(Corpus)

Term Frequency

The	Matrix	is	nothing	but	an	advertising	gimmick
40	5	50	12	20	45	3	2

Document Frequency

The	Matrix	is	nothing	but	an	advertising	gimmick
2000	190	2300	500	1200	3000	52	12

$$TFIDF_i = TF_i * \log\frac{N}{DF_i}$$

TF_i = 개별 문서에서의 단어 i 빈도

DF_i = 단어 i를 가지고 있는 문서 개수

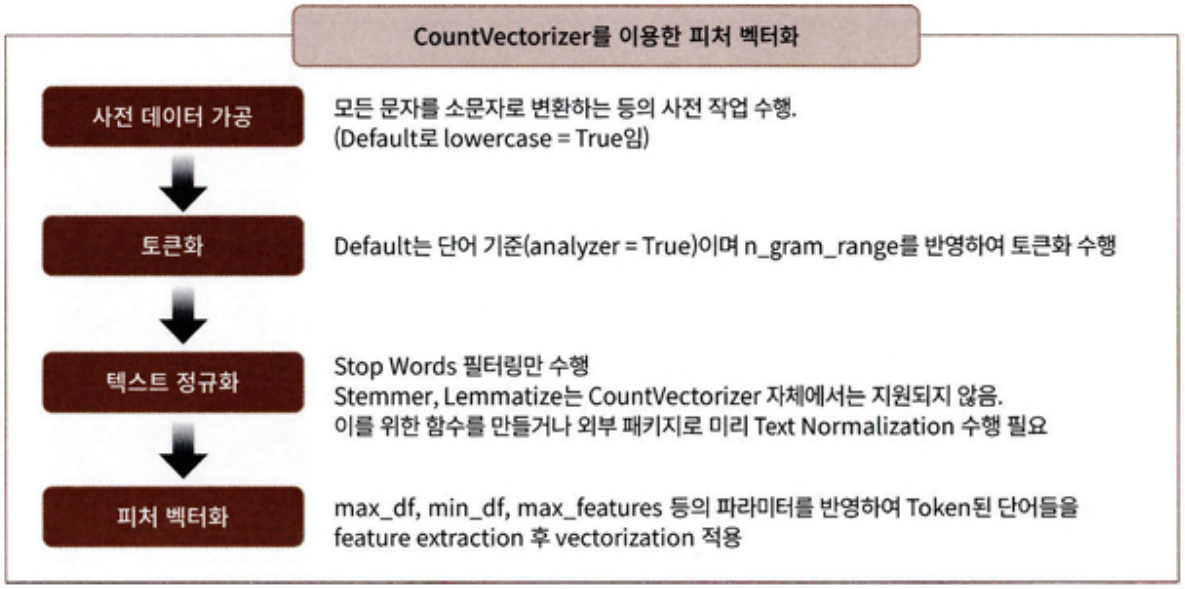
N = 전체 문서 개수

사이킷런 → CountVectorizer, TfidfVectorizer

✅ **CountVectorizer**: 카운트 기반 벡터화 구현한 클래스. 텍스트 전처리도 함께 수행

파라미터 명	파라미터 설명
max_df	전체 문서에 걸쳐서 너무 높은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터입니다. 너무 높은 빈도수를 가지는 단어는 스톱 워드와 비슷한 문법적인 특성으로 반복적인 단어일 가능성이 높기에 이를 제거하기 위해 사용됩니다. max_df = 100과 같이 정수 값을 가지면 전체 문서에 걸쳐 100개 이하로 나타나는 단어만 피처로 추출합니다. Max_df = 0.95와 같이 부동소수점 값(0.0 ~ 1.0)을 가지면 전체 문서에 걸쳐 빈도수 0~95%까지의 단어만 피처로 추출하고 나머지 상위 5%는 피처로 추출하지 않습니다.
min_df	전체 문서에 걸쳐서 너무 낮은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터입니다. 수백~수천 개의 전체 문서에서 특정 단어가 min_df에 설정된 값보다 적은 빈도수를 가진다면 이 단어는 크게 중요하지 않거나 가비지(garbage)성 단어일 확률이 높습니다. min_df = 2와 같이 정수 값을 가지면 전체 문서에 걸쳐서 2번 이하로 나타나는 단어는 피처로 추출하지 않습니다. min_df = 0.02와 같이 부동소수점 값(0.0 ~ 1.0)을 가지면 전체 문서에 걸쳐서 하위 2% 이하의 빈도수를 가지는 단어는 피처로 추출하지 않습니다.
max_features	추출하는 피처의 개수를 제한하며 정수로 값을 지정합니다. 가령 max_features = 2000으로 지정할 경우 가장 높은 빈도를 가지는 단어 순으로 정렬해 2000개까지만 피처로 추출합니다.
stop_words	'english'로 지정하면 영어의 스톱 워드로 지정된 단어는 추출에서 제외합니다.
n_gram_range	Bag of Words 모델의 단어 순서를 어느 정도 보강하기 위한 n_gram 범위를 설정합니다. 튜플 형태로 (범위 최솟값, 범위 최댓값)을 지정합니다. 예를 들어 (1, 1)로 지정하면 토큰화된 단어를 1개씩 피처로 추출합니다. (1, 2)로 지정하면 토큰화된 단어를 1개씩(minimum 1), 그리고 순서대로 2개씩(maximum 2) 묶어서 피처로 추출합니다.
analyzer	피처 추출을 수행한 단위를 지정합니다. 당연히 디폴트는 'word'입니다. Word가 아니라 character의 특정 범위를 피처로 만드는 특정한 경우 등을 적용할 때 사용됩니다.
token_pattern	토큰화를 수행하는 정규 표현식 패턴을 지정합니다. 디폴트 값은 '\b\w\w+\b로, 공백 또는 개행 문자 등으로 구분된 단어 분리자(\b) 사이의 2문자(문자 또는 숫자, 즉 영숫자) 이상의 단어(word)를 토큰으로 분리합니다. analyzer= 'word'로 설정했을 때만 변경 가능하나 디폴트 값을 변경할 경우는 거의 발생하지 않습니다.
tokenizer	토큰화를 별도의 커스텀 함수로 이용시 적용합니다. 일반적으로 CountTokenizer 클래스에서 어근 변환 시 이를 수행하는 별도의 함수를 tokenizer 파라미터에 적용하면 됩니다.

CountVectorizer 클래스를 이용한 카운트 기반의 피처 벡터화



✅ TfidfVectorizer도 파라미터 & 변환 방법 동일

BOW 벡터화를 위한 희소 행렬

사이킷런의 CountVectorizer/TfidfVectorizer로 텍스트를 피처 단위로 벡터화해 변환
→ CSR 형태의 희소 행렬 반환

✏️ BOW 형태를 가진 언어 모델의 피처 벡터화는 대부분 희소 행렬

수십만 개의 칼럼 →

수천~수만 개 레코드

	단어 1	단어 2	단어 3	....	단어 1000	....	단어 2000	....	단어 10000	....	단어 20000	....	단어 100000
문서1	1	2	2	0	0	0	0	0	0	0	0	0	0
문서2	0	0	1	0	0	1	0	0	1	0	0	0	1
문서 ...	....	....	....	....	....	....	....	....	....	....	....	....	....
문서 10000	0	1	3	0	0	0	0	0	0	0	0	0	0

BOW의 Vectorization 모델은 너무 많은 0값이 메모리 공간에 할당되어 많은 메모리 공간이 필요하며
연산 시에도 데이터 액세스를 위한 많은 시간이 소모됩니다.

→ 이러한 희소 행렬을 물리적으로 적은 메모리 공간을 차지할 수 있도록 변환해야 함 — COO, CSR 형식

희소 행렬 - COO 형식

Coordinate 형식: 0이 아닌 데이터만 별도의 데이터 array에 저장하고, 그 데이터가 가리키는 행과 열의 위치를 별도의 배열로 저장하는 방식

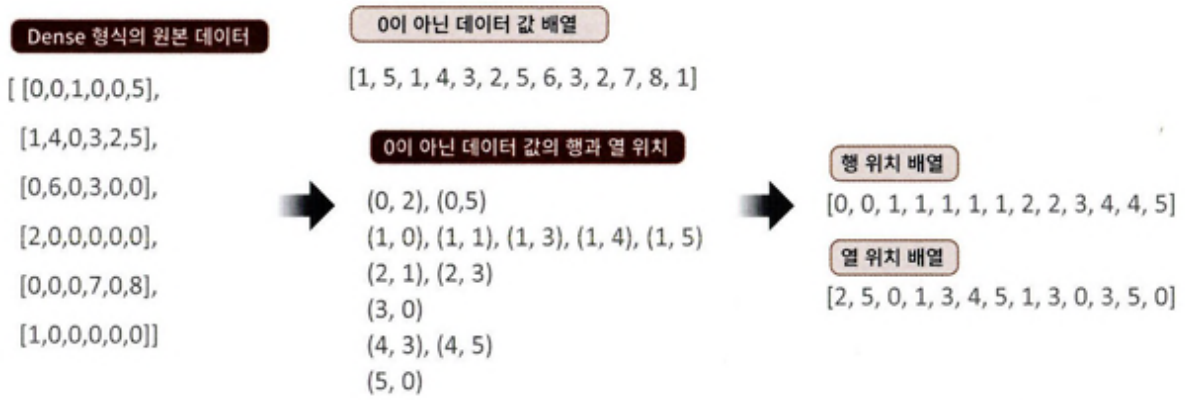
ex) [[3,0,1], [0,2,0]] → [3,1,2]의 좌표를 배열로 저장

** 사이파이의 sparse 패키지 사용

희소 행렬 - CSR 형식 - scipy.sparse.csr_matrix

Compressed Sparse Row 형식: COO 형식은 행과 열의 위치를 나타내기 위해 반복적인 위치 데이터를 사용해야 하는 문제점을 해결한 방식

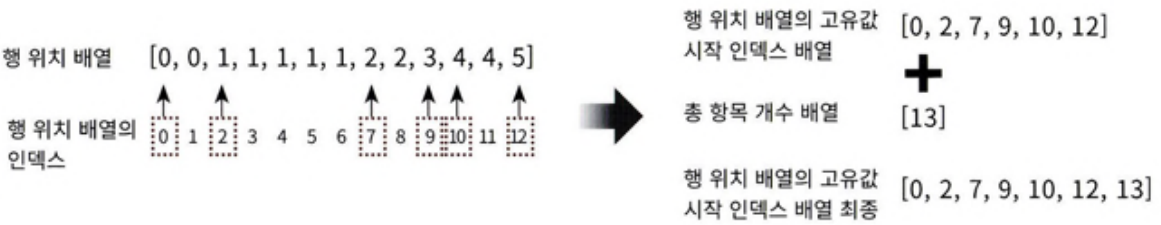
⚠️ COO 변환 형식의 문제점



행 위치 배열에서 순차적인 같은 값이 반복적으로 나타나고 있음

→ 각 값의 시작 위치만 표기 = 위치의 위치를 표기

⇒ **CSR**: 행 위치 배열 내에 있는 고유한 값의 시작 위치만 다시 별도의 위치 배열로 가지는 변환 방식



행 값의 시작 인덱스 배열 + 총 항목 개수 배열 → 최종 CSR 배열

→ 최종 변환된 배열로 다시 행 위치 배열을 다시 만들 수 있음. COO 방식보다 메모리가 적게 들고 빠른 연산 가능

감성 분석 (Sentiment Analysis)

문서의 주관적인 감성/의견/감정/기분 등을 파악하기 위한 방법
— SNS, 여론조사, 온라인 리뷰, 피드백 등 다양한 분야에서 활용되고 있음

문서 내 텍스트가 나타내는 여러 주관적 단어와 문맥을 기반으로 감성(Sentiment) 수치를 계산하는 방법을 이용

✅ 긍정 감성 지수 & 부정 감성 지수 합산 → 긍정 감성 또는 부정 감성을 결정

- 지도학습
  : 학습 데이터 & 타깃 레이블 값을 기반으로 감성 분석 학습 수행
  → 이를 기반으로 다른 데이터의 감성 분석 예측
- 비지도학습
  : 'Lexicon' 감성 어휘 사전 이용 - Lexicon은 감성 분석을 위한 용어와 문맥에 대한 다양한 정보를 담고 있음. 이를 이용해 문서의 긍정 부정 감성 여부 판단

비지도학습 기반 감성 분석

Lexicon을 기반으로 하는 것

감성 사전

- **감성 지수(Polarity score)** = 긍정 감성 또는 부정 감성의 정도를 의미하는 수치를 가지고 있음
  - 단어의 위치, 주변 단어, 문맥, POS(Part of Speech) 등을 참고해 결정
- NLTK 패키지의 **WordNet**: 시맨틱 분석을 제공하는 방대한 영어 어휘 사전
  - 시맨틱(semantic): '문맥상 의미'
  - 어휘의 시맨틱 정보 제공 - 각각의 품사로 구성된 개별 단어를 Synset(Sets of cognitive synonyms - 단어가 가지는 문맥)라는 개념을 이용해 표현
  - 예측 성능이 그리 좋지 못함
- 이외의 감성 사전 종류
  - **SentiWordNet**

    : 감성 단어 전용의 WordNet 구성한 것. WordNet의 Synset 개념을 감성 분석에 적용한 것
    Synset별로 3가지 감성 점수 할당 = 긍정 감성 지수, 부정 감성 지수, 객관성 지수
    - 긍정 감성 지수: 해당 단어가 감성적으로 얼마나 긍정적인가

        - 부정 감성 지수: 해당 단어가 감성적으로 얼마나 부정적인가
        - 객관성 지수: 해당 단어가 감성과 관계없이 얼마나 객관적인가

      → 이 지수들을 합산하여 감성이 긍정인지 부정인지 결정

- **VADER**

  : 주로 소셜 미디어의 텍스트에 대한 감성 분석을 제공하기 위한 패키지

  - 뛰어난 감성 분석 결과
  - 비교적 빠른 수행 시간 보장 → 대용량 👍

- Pattern