# 12주차_8장 Part1. 텍스트 분석 (8.1~8.3, 8.5)

| | |
|---|---|
| 상태 | 진행 중 |
| 마감일 | @11/24/2025 |

# 1. 텍스트 분석 이해

:비정형인 텍스트 분석 ↔ ML 모델의 주어진 정형 데이터 분석

- 어떻게 피처 형태로 추출할 것인가?
- 어떻게 추출한 피처에 의미 있는 값을 부여하는가?

피처 벡터화/피처 추출 : 텍스트를 word 기반 다수의 피처 추출 → 피처에 단어 빈도수 등 숫자 값 부여 → 텍스트의 단어의 조합인 벡터값 표현, 변환

ex) BOW, Word2Vec

## 01-1 NLP vs. 텍스트 분석

NLP : 머신의 인간 언어 이해&해석

텍스트 분석 : 비정형 텍스트 속 의미 o 정보 추출

▼ 텍스트 분류

문서가 특정 카테고리에 속하는 것 예측

▼ 감성 분석

텍스트의 주관적 요소 분석

▼ 텍스트 요약

텍스트 내 중요 주제와 사상 추출

▼ 텍스트 군집화

비슷한 유형의 문서에 대해 군집화

## 01-2 텍스트 분석 수행 프로세스

1. **텍스트 사전 분리작업**
    - 피처 변환 전 클렌징
    - 대/소문자 변경
    - 특수문자 제거
    - 단어의 토큰화
    - 의미 없는 단어=Stop word 제거
    - 어근 추출=Stemming/Lemmatization
2. **피처 벡터화/추출**

    클렌징된 텍스트에서 피처 추출, 여기에 벡터 값 할당

    ▼ Bow
        - Count 기반 벡터화
        - TF-IDF 기반 벡터화

    ▼ Words2Vec
3. **ML 모델 수립 및 학습/예측/평가**

## 01-3 파이썬 기반의 NLP, 텍스트 분석 패키지

▼ NLTK

파이썬 대표 NLP 패키지

속도 ↓로 대량 데이터에는 x

▼ Gensim

토픽 모델링 분야 사용

▼ SpaCy

뛰어난 성능으로 최근 주목

+사이킷런은 특화 라이브러리는 갖고 있지 않지만, 피처 변환 등 편리한 기능 때문에 위 패키지와 함께 사용됨

# 2. 텍스트 사전 준비 작업(텍스트 전처리)-텍스트 정규화

## 02-1 클렌징

: 분석에 방해가 되는 것들 사전에 제거

ex) 불필요한 문자, 기호, HTML/XML 태그

## 02-2 텍스트 토큰화

▼ 문장 토큰화 Sentence tokenization

: 문서에서 문장 분리

- 각 문장이 가지는 시맨틱적 의미가 중요할 때 사용
- 마침표(.), 개행문자(\n) 등 문장의 마지막을 뜻하는 기호에 따라 분리하는 것이 일반적
- 정규 표현식에 따른 문장 토큰화도 가능 -
  - NLTK `sent_tokenize` : 각각의 문장으로 구성된 list 객체 반환

▼ 단어 토큰화

: 문장에서 단어 분리

- Bag of Word 등 단어 순서 중요 X 일때 단어 토큰화만으로도 충분
- 공백, 콤마(,), 마침표(.), 개행문자(\n) 등으로 단어를 분리
- 정규 표현식으로 다양한 유형 토큰화도 가능
  - NLTK `word_tokenize` :

+(P) 문장을 단어별로 토큰화할 경우 문맥적 의미 무시

(S) **n-gram**

: 연속된 n개의 단어를 하나의 토큰화 단위로 분리

ex) 2-gram(biagram) - 연속적으로 2개의 단어들을 순차적으로 이동하면서 단어들을 토큰화

## 02-3 스톱 워드 제거

Stop word : 분석에 큰 의미 X 단어

ex) is, the, a, will 등 필수 문법 요소지만 문맥적 의미 X 단어들

- 사전에 제거 X 시 빈번함으로 인해 오히려 중요한 단어로 인지되는 문제 발생
- 언어별로 스톱 워드 목록화돼 있음

  ex) `nltk.corpus.stopword.words('english')`

## 02-4 Stemming과 Lemmatization

: 문법적 또는 의미적으로 변화하는 단어의 원형을 찾는 것

ex) workd, works, working → 원형 work

▼ Lemmatization

- Stemming보다 정교
- 의미론적 기반에서 단어 원형 찾음
- 시간 ↑
- NLTK 의 `WorldNetLemmatizer`
  - 단어의 품사 입력 필요 - 동사 'v', 형용사 'a'
  - ex) `print(lemma.lemmatize('fancier','a'),lemma.lemmatize('fanciest','a'))`

▼ Stemming

- 원형 단어로 변환 시 일반적 방법/더 단순화된 방법 적용 → 원래 단어에서 **일부 철자가 훼손된** 어근 단어 추출하는 경향 있음
  - 진행형/3인칭단수/과거동사/비교/최상급 등 더 단순하게 원형 단어 찾음
- NLTK 의 `Porter`, `Lancaster`, `Snowball Stemmer`
  - `Lancaster` : 필요한 Stemmer 객체 생성 후 이 객체의 `stem('단어')` 메서드 호출 → '단어'의 Stemming 可

# 3. Bag of Words - BOW

: 문서의 모든 단어를 **문맥/순서 무시**하고 **일괄적 빈도 값** 부여해 피처 값 추출하는 모델

예시 알고리즘 - 문장 1&문장2

1. 문장 두개 의 모든 단어에서 중복 제거하고, 각 단어를 칼럼 형태로 나열 후 고유의 인덱스 부여
2. 개별 문장에서 해당 단어의 빈도를 인덱스에 기제

- 장점
  - 쉽고 빠른 구축
    - 단순히 횟수 기반이지만, 예상보다 문서의 특징 잘 알 수 있음
- 단점
  - 문맥 의미 반영 부족 - 단어 순서 고려 x여서
  - 희소 행렬 문제
    - 단어 개수가 수십~수십만개인데 행렬 대부분이 0으로 채워지는 희소 행렬 (↔ 밀집 행렬)
    - → 시간과 성능 문제

# 03-1 BOW 피처 벡터화

: 텍스트를 **특정 의미를 가지는 숫자**형 값인 벡터 값으로 변환하는 것

-머신러닝 알고리즘은 숫자형 피처를 input받아서 텍스트 데이터는 바로 입력할 수 없기 때문

- 모든 문서에서 / 모든 단어를 칼럼 형태로 나열하고 / 단어 횟수 or 정규화된 빈도를 값으로 부여해/ 데이터 세트 모델로 변경
- M개 문서 & 모두 N개 단어 ⇒ M X N 개 단어 피처로 이뤄진 행렬 구성

**▼ 카운트 기반의 벡터화 방식**

- 단어 피처에 횟수=count를 값으로 부여하는 것
- 높을수록 중요한 단어로 인식
- P : 문서의 특징보다는 언어 특성상 자주 사용될 수 밖에 없는 단어까지 높은 값 부여

**▼ TF-IDF 기반의 벡터화 방식 (Term frequency - Inverse Document frequency)**

- 카운트 기반의 벡터화 문제 보완
- **개별 문서**에서 자주 나타나는 단어에 높은 **가중치** + **모든 문서**에서 자주 나타는 단어에 **페널티**
- 문서가 많고 문서마다 텍스트가 길 때 사용하면 좋은 예측 성능

# 03-2 사이킷런의 Count 및 TF-IDF 벡터화 구현 : CountVectorizer, TfidfVectorizer

- 사이킷런 `CountVectorizer` 클래스 : 카운트 기반 벡터화 구현
    - 피처 벡터화 + 소문자 일괄 변환 + 토큰화 + 스톱 워드 필터링

▼ 주요 입력 파라미터

- `max_df` : 전체 문서 걸쳐 너무 **높은** 빈도수 가지는 단어 피처 제외용. 지정된 값 이하로 나타나는 단어만 피처로 추출
- `min_df` : 전체 문서 걸쳐 너무 **낮은** 빈도수 가지는 단어 피처 제외용. 지정된 값 이하로 나타는 단어는 피처로 추출하지 X
- `max_features` : 추출하는 피처 개수 제한 오름차순으로
- `stop_words` : 해당 언어의 스톱 워드로 지정된 단어는 추출 제외
- `n_gram_range` : BOW의 단어 순서 보강 위한 n_gram 범위 설정. 튜플 형태 (범위 최솟값, 범위 최댓값) 입력
- `analyzer` : 피처 추출을 수행한 단위 지정. 디폴드 'word'. word 아닌 character의 특정 범위를 피처로 만드는 특수한 경우 적용 시 사용
- `token_pattern` : 토큰화 수행 정규 표현식 패턴 지정. 디폴트 '\b\w\w+\b'  - 공백 or 개행 문자 등으로 구분된 단어 분리자 \b 사이의 2문자 이상의 단어를 토큰으로 분리
- `tokenizer` : 토큰화를 별도의 커스텀 함수로 이용시 적용

# 03-3 BOW 벡터화를 위한 희소 행렬

- 단어가 많기 때문에 대규모 행렬 생성되더라도 대부분의 값이 0 = 희소 행렬.
- BOW 기반 피처 벡터화는 대부분 희소 행렬
- 메모리 공간과 시간 문제 때문에 물리적으로 적은 메모리 공간 차지할 수 있도록 변환해야 함

# 03-4 희소 행렬 처리 - COO 형식

: 0이 아닌 데이터만 별도의 배열에 저장하고, 그 데이터가 가리키는 행/열의 위치를 별도의 배열로 저장

ex) [  [ 3, 0, 1],    → 0이 아닌 데이터 [3,1,2] → 위치 (0,0), (0,2), (1,1) → 별도의 배열 저장 시 행 [0,0,1], 열 [0,2,1]

[0, 2, 0] ]

## 03-5 희소 행렬 처리 - CSR  형식

+COO문제 : 행/열 위치 나타내기 위해 반복적인 위치 데이터 사용해야 함

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

# 5. 감성 분석

## 05-1 감성 분석 소개

문서의 주관적인 감성을 파악하기 위한 방법

- 텍스트가 나타내는 여러 주관적 단어 + 문맥 기반 감성 수치 계산
- 긍정 감성 지수 + 부정 감성 지수 ⇒ 긍정 감성/부정 감성 결정

▼ 지도학습 방식

학습 수행 후 이를 기반으로 다른 데이터의 감성 분석 예측

▼ 비지도 학습 방식

‘Lexicon’이라는 감성 어휘 사전 이용해 문서의 긍/부정적 감성 여부 판단

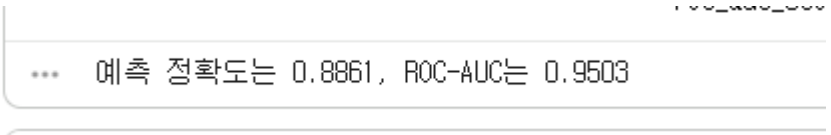
## 05-2 지도학습 기반 감성 분석 실습 - IMDB 영화평

데이터 설명

- id : 각 데이터 id
- sentiment : 영화평의 감성 결과 값(Target). 0=긍정, 1 = 부정
- review : 영화평 텍스트

1. 정규 표현식으로 HTML 형식의 <br /> 태그, 숫자와 특수문자 공란 변경
2. sentiment 칼럼 별도로 추출해 결정 값 데이터 세트 생성 +  원본 데이터에서 id/sentiment 칼럼 삭제해 피처 데이터 세트 생성 , train_test_split()으로 학습/테스트 데이터 세트 분리
3. review 텍스트를 피처 벡터화한 후 ML 분류 알고리즘으로 예측 성능 추정

결과 1_ Count 벡터화 :



결과2_TF-IDF 벡터화 :


조금 더 성능이 나아짐

# 05-3 비지도학습 기반 감성 분석 소개

: Lexicon - 긍정 감성과 부정 감성의 정도를 의미하는 수치인 감성지수Polarity score 가지고 있음

← by 단어의 위치, 주변 단어, 문맥, POS(part of speech)

NLTK 패키지에서 대표적으로 구현돼 있음, 그 속 Lexicon 모듈

NLTK 패키지의 WordNet : 다양한 상황에서 같은 어휘라고 다르게 사용되는 어휘의 시멘틱 정보 제공

- 이를 위해 각 품사로 구성된 개별 단어를 Synset 이라는 개념으로 표현
  - set of cognitive synonyms : 단순한 하나의 언어가 아니라 그 단어가 가지는 문맥&시맨틱 정보를 제공하는 WordNet의 핵심 개념

NLTK 외 감성 사전:

▼ SentiWordNet :

- WordNet의 synset 개념을 감성 분석에 적용. synset별로 3가지 감성 점수 할당 (객관, 긍정, 부정)
- 긍정 감성 지수 + 부정 감성 지수 ⇒ 최종 감성 지수 ⇒ 감성의 긍/부정 여부 판단

▼ VADER:

- 주로 SNS의 텍스트에 대한 감성 분석
- 빠른 시간 → 대용량 텍스트 데이터에 용이한 패키지

▼ Pattern:

- 예측 성능 면에서는 가장 주목받지만 파이썬 3.X 버전 아직 호환 x

# 05-4 SentiWorldNet을 이용한 감성 분석

- synset 객체의 여러 속성 살펴보기 - ‘present’의 예시
  - (우리말로 하면 품사)
  - 정의, 부명제 등으로 시맨틱적 요소 표현 可

```
##### Synset name :  present.n.01 #####
POS : noun.time
Definition: the period of time that is happening now; any continuous stretch of time including the moment of speech
Lemmas: ['present', 'nowadays']
##### Synset name :  present.n.02 #####
POS : noun.possession
Definition: something presented as a gift
Lemmas: ['present']
##### Synset name :  present.n.03 #####
POS : noun.communication
Definition: a verb tense that expresses actions or states at the time of speaking
Lemmas: ['present', 'present_tense']
```

    - 명사지만 다른 의미인 단어들
    - 동사인 단어들

  → 하나의 단어가 가질 수 있는 여러 시맨틱 정보를 개별 클래스로 나타낸 것
- WordNet과 synset 객체로 어휘들 간 유사도 살펴보기
  - synset 객체의 `path_similarity()` 메서드 사용

| | tree | lion | tiger | cat | dog |
|---|---|---|---|---|---|
| **tree** | 1.00 | 0.07 | 0.07 | 0.08 | 0.12 |
| **lion** | 0.07 | 1.00 | 0.33 | 0.25 | 0.17 |
| **tiger** | 0.07 | 0.33 | 1.00 | 0.25 | 0.17 |
| **cat** | 0.08 | 0.25 | 0.25 | 1.00 | 0.20 |
| **dog** | 0.12 | 0.17 | 0.17 | 0.20 | 1.00 |

  lion은 tree와의 유사도가 0.07로 가장 적고, tiger와의 유사도가 0.33으로 가장 큼
- SentiWordNet의 Senti_Synset 클래스로 단어의 감성 지수 알아보기 - ‘father’, ‘fabulous’의 예시

```
father 긍정감성 지수:  0.0
father 부정감성 지수:  0.0
father 객관성 지수:  1.0


fabulous 긍정감성 지수:  0.875
fabulous 부정감성 지수:  0.125
```

father는 객관적 단어, fabulous는 긍정 감성 단어인 것 알 수 있음

# 05-5 VADER를 이용한 감성 분석

: SNS용 감성 분석 용도로 만들어진 룰 기반의 Lexicon

- SentimentIntensityAnalyzer 클래스로 감성 분석 제공

1. SentimentIntensityAnalyzer 객체 생성
2. 문서별로 `polarity_scores()` 메서드 호출해 감성 점수 획득
    - `polarity_scores()` : 딕셔너리 형태의 감성 점수 반환
        - ‘neg’ : 부정감성
        - ‘neu’ : 중립감성
        - ‘pos’ : 긍정감성
    - compound : ‘neg’, ‘neu’, ‘pos’를 조합해 -1~1 사이의 감성 지수 표현
        - coumpound score 기반으로 부정 감성 or 긍정 감성 여부 판단
        - 0.1 이상 : 긍정 감성, but threshold 조정 가능
3. 해당 문서의 감성 점수가 threshold 이상이면 긍정, 아니면 부정으로 판단