8.0 - 8.1 텍스트 분석 개요 및 이해

초급 1팀 정현빈

EWHA EURON

#텍스트 분석 개요

텍스트 분석

비정형 데이터인 텍스트를 분석하는 것.

* 머신러닝 알고리즘은 숫자형의 피처 기반 데이터만 입력 받을 수 있기 때문에 비정형 텍스트 데이터를 피처 형태로 추출하고, 추출된 피처에 의미 있는 값을 부여하는 것이 중요

NLP vs 텍스트 분석

- NLP : 머신이 인간의 언어를 이해하고 해석하는 데 중점

- 텍스트 분석 : 비정형 텍스트에서 의미 있는 정보를 추출하는 것에 중점

#텍스트 분석 개요

머신러닝에 기반한 텍스트 분석 기술 영역

영역	설명
텍스트 분류	문서가 특정 분류 또는 카테고리에 속하는 것을 예측하는 기법 예) 특정 신문 기사 내용이 연애/정치/사회/문화 중 어떤 카테고리에 속하는지 자동으로 분류, 스팸 메일 검출
감성 분석	텍스트에서 나타나는 감정/판단/믿음/의견/기분 등의 주관적인 요소를 분석하는 기법 예) 소셜 미디어 감정 분석, 영화나 제품에 대한 긍정 또는 리뷰, 여론조사 의견 분석
텍스트 요약	텍스트 내에서 중요한 주제나 중심 사상을 추출하는 기법 예) 토픽 모델링
텍스트 군집화와 유사도 측정	비슷한 유형의 문서에 대해 군집화를 수행하는 기법 텍스트 분류를 비지도학습으로 수행하는 방법의 일환으로 사용될 수 있음

#텍스트 분석 개요

텍스트 분석 활용

실무	연구
스팸 필터링	사회동향 분석
이슈 검출/트래킹	이슈 트래킹
정보 검색	온라인 행동 분석
자살률 예측	연구분야 탐색
주가 예측	질병관계 예측
소비자 인식 조사	정책전략 수립
경쟁사 분석	

#텍스트 분석 이해

텍스트 분석 수행 프로세스

step 1. 텍스트 사전 준비작업(텍스트 전처리) :

텍스트를 피처로 만들기 전에 미리 클렌징, 대/소문자 변경, 특수문자 삭제 등의 클렌징 잡업, 단어 등의 토큰화 작업,

의미 없는 단어 제거 작업, 어근 추출 등의 텍스트 정규화 작업을 수행하는 것

step 2. 피처 벡터화/추출 :

사전 준비 작업으로 가공된 텍스트에서 피처를 추출하고 여기에 벡터 값을 할당. 대표적인 방법에는 BOW와

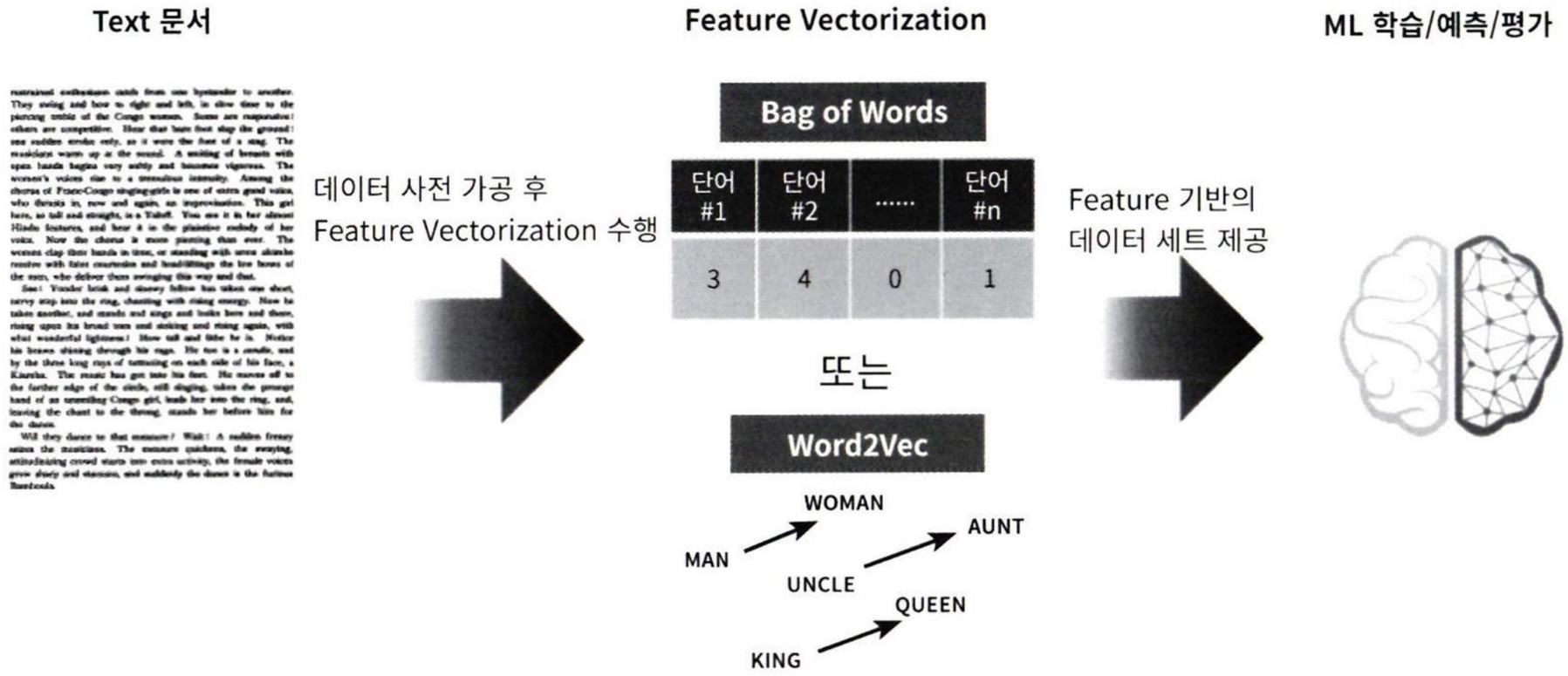
Word2Vec이 있음

step3. ML 모델 수립 및 학습 / 예측 / 평가 :

피처 벡터화된 데이터 세트에 ML 모델 적용해 학습 / 예측 및 평가 수행

#텍스트 분석 이해

텍스트 분석 수행 프로세스



#텍스트 분석 이해

텍스트 분석 수행 프로세스

1. 데이터 준비

API 호출	Open API(Rest API)를 제공하는 서비스에 Open API를 호출
웹 크롤링	웹 상에 존재하는 콘텐츠를 수집하는 작업 (파싱 및 Selenium 통한 조작)
기타	오프라인 수집, 이미지 및 파일에서 추출, etc..

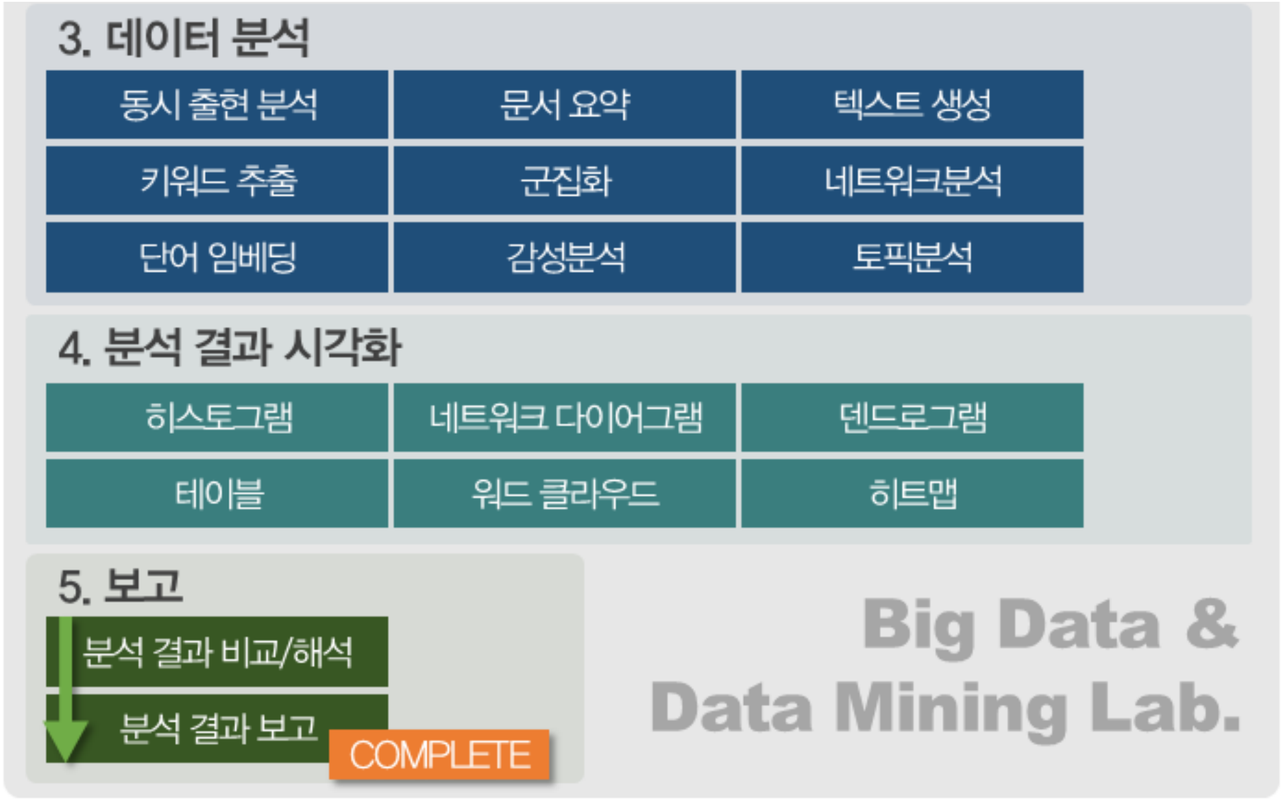
2. 데이터 전처리 ★

텍스트 마이닝 과정 중에서 시간이 가장 많이 소요되는 작업

데이터 정규화	표현 방법이 다른 단어들을 통합 Ex) 예시입니닿ㅎㅎ → 예시입니다 ㅎㅎ
데이터 분리	데이터를 특성에 따라 분리할 필요가 있을 경우에 진행
형태소 분석 (토큰화)	일정한 의미가 있는 가장 작은 말의 단위로 변환 (품사 태깅)
개체명 인식	이름을 가진 개체(named entity)로 인식 (Ex. 소희 – 사람, 동국대 – 조직)
원형 복원	형용사/동사의 표현형 -> 원형 (Ex. 쉬고싶다, 쉬고싶은 -> 쉬다) 어간/어미 -> 표현형으로 활용 (Ex. 보다, 보니, 보고 -> 보-)
불용어 제거	조사, 접미사 – 나, 너, 은, 는, 이, 가, 하다, 합니다 등
단어 빈도 분석	불용어 및 빈출어의 제거 여부 & 필요한 단어들의 올바른 추출 확인

#텍스트 분석 이해

텍스트 분석 수행 프로세스



#텍스트 분석 이해

NLTK

- 파이썬의 가장 대표적인 NLP 패키지

- 방대한 데이터 세트와 서브 모듈을 가지고 있음

- 수행 속도 측면에서 아쉬운 부분이 있어 실제 대량의 데이터 기반에서는 제대로 활용되지 못함

- 주요 기능으로 말뭉치, 토큰 생성, 형태소 분석, 품사 태깅이 있음

#텍스트 분석 이해

KoNLPy

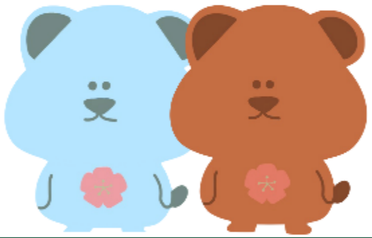
- 한국어 자연어 처리를 위한 형태소 분석기 패키지
- 파이썬에서 활용 가능한 오픈 소스 형태소 분석기
- 기존에 공개된 여러 분석기를 한 번에 설치 및 동일한 방법으로 사용 가능하도록 해줌
- 주요 기능으로 형태소 분석 morphs( ), 품사 태깅 pos( ) 가 있음

#텍스트 분석 이해

Gensim

- 토픽 모델링 분야에서 가장 두각을 나타내는 패키지
- Word2Vec 라이브러리
- 효율성 및 확장 가능성으로 인해 폭넓게 사용되고 있음
- 주요 기능으로 임베딩(Word2Vec), 토픽 모델링, LDA가 있음

8.2 텍스트 정규화



초급 멤버 김선향

EWHA EURON

#8.2 텍스트 정규화

1. 클렌징
2. 텍스트 토큰화
3. 필터링/스톱 워드 제거/철자 수정
4. Stemming
5. Lemmatization

#8.2 텍스트 정규화

▼ 클렌징 🛁🫧

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

#8.2 텍스트 정규화

▼ 텍스트 토큰화

문장 토큰화 (`sent_tokenize`)

- 일반적으로 문장 토큰화는 각 문장이 가지는 시맨틱적인 의미가 중요한 요소로 작용할 때 사용
- 문장의 마침표(.), 개행문자(/n) 등 문장의 마지막을 뜻하는 기호에 따라 분리하는 과정
- 정규 표현식에 따른 문장 토큰화도 가능함

#8.2 텍스트 정규화

```
from nltk import sent_tokenize
import nltk
nltk.download('punkt')

text_sample = 'The Matrix is everywhere its all around us, here even in this room. ₩
              You can see it out your window or on your television. ₩
              You feel it when you go to work, or go to church or pay your taxes.'
sentences = sent_tokenize(text=text_sample)
print(type(sentences),len(sentences))
print(sentences)
```

```
<class 'list'> 3
['The Matrix is everywhere its all around us, here even in this room.', 'You can see it out yo
ur window or on your television.', 'You feel it when you go to work, or go to church or pay yo
ur taxes.']
```

#8.2 텍스트 정규화

단어 토큰화 (`word_tokenize`)

- 문장을 단어로 토큰화하는 것기본적으로 공백, 콤마(,), 마침표(.), 개행문자(\n) 등으로 단어를 분리
- 정규 표현식을 이용하여 다양한 유형으로 토큰화 수행 가능

#8.2 텍스트 정규화

```
from nltk import word_tokenize

sentence = "The Matrix is everywhere its all around us, here even in this room."
words = word_tokenize(sentence)
print(type(words), len(words))
print(words)
```

```
<class 'list'> 15
['The', 'Matrix', 'is', 'everywhere', 'its', 'all', 'around', 'us', ',', 'here', 'even', 'in',
'this', 'room', '.']
```

#8.2 텍스트 정규화

문장 토큰화 + 단어 토큰화

```
from nltk import word_tokenize, sent_tokenize

#여러개의 문장으로 된 입력 데이터를 문장별로 단어 토큰화 만드는 함수 생성
def tokenize_text(text):

    # 문장별로 분리 토큰
    sentences = sent_tokenize(text)
    # 분리된 문장별 단어 토큰화
    word_tokens = [word_tokenize(sentence) for sentence in sentences]
    return word_tokens

#여러 문장들에 대해 문장별 단어 토큰화 수행.
word_tokens = tokenize_text(text_sample)
print(type(word_tokens),len(word_tokens))
print(word_tokens)
```

```
<class 'list'> 3
[['The', 'Matrix', 'is', 'everywhere', 'its', 'all', 'around', 'us', ',', 'here', 'even', 'in', 'this', 'room', '.'], ['You',
'can', 'see', 'it', 'out', 'your', 'window', 'or', 'on', 'your', 'television', '.'], ['You', 'feel', 'it', 'when', 'you', 'g
o', 'to', 'work', ',', 'or', 'go', 'to', 'church', 'or', 'pay', 'your', 'taxes', '.']]
```

#8.2 텍스트 정규화

⚠ BUT !

P : 문장을 단어별로 토큰화 할 경우, 문맥적인 의미는 무시될 수밖에 없음.

S : `n-gram`

`n-gram` : 연속된 n 개의 단어를 하나의 토큰화 단위로 분리하는 것

2-gram

"Agent Smith knocks the door" → (Agent, Smith) , (Smith, knocks), (knocks, the), (the, door)

#8.2 텍스트 정규화

▼ 스톱 워드 제거

- `Stop Word`
  - 분석에 큰 의미가 없는 단어를 지칭
    - (예 - is, a, the, will 등 문장을 구성하는 필수 문법 요소이지만 문맥적으로는 큰 의미가 없는 단어)
  - 이러한 단어는 사전에 제거하지 않으면 그 빈번함으로 인해 오히려 중요한 단어로 인지될 수 있음
  - NLTK 에서는 언어별 스톱 워드 목록을 제공

#8.2 텍스트 정규화

```
import nltk
nltk.download('stopwords')
```

```
print('영어 stop words 갯수:',len(nltk.corpus.stopwords.words('english')))
print(nltk.corpus.stopwords.words('english')[:20])
```

```
영어 stop words 갯수: 179
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his']
```

- NLTK 에서는 언어별 스톱 워드 목록을 제공

#8.2 텍스트 정규화

스톱워드 제거

```
import nltk

stopwords = nltk.corpus.stopwords.words('english')
all_tokens = []
# 위 예제의 3개의 문장별로 얻은 word_tokens list 에 대해 stop word 제거 Loop
for sentence in word_tokens:
    filtered_words=[]
    # 개별 문장별로 tokenize된 sentence list에 대해 stop word 제거 Loop
    for word in sentence:
        #소문자로 모두 변환합니다.
        word = word.lower()
        # tokenize 된 개별 word가 stop words 들의 단어에 포함되지 않으면 word_tokens에 추가
        if word not in stopwords:
            filtered_words.append(word)
    all_tokens.append(filtered_words)

print(all_tokens)
```

【Output】

```
[['matrix', 'everywhere', 'around', 'us', ',', 'even', 'room', '.'], ['see', 'window', 'television',
'.'], ['feel', 'go', 'work', ',', 'go', 'church', 'pay', 'taxes', '.']]
```

#8.2 텍스트 정규화

▼ Stemming, Lemmatization

Stemming 과 **Lemmatization** 은 **문법적 또는 의미적으로 변화하는 단어의 원형을 찾는 과정**

- Stemming
  - 원형 단어를 변환 시 일반적인 방법을 적용하거나, 더 단순화된 방법을 적용
  - 원래 단어에서 일부 철자가 훼손된 어근 단어를 추출하는 경향이 있음
- Lemmatization
  - Stemming 에 비해 더 정교하며, 의미론적인 기반에서 단어의 원형을 찾음
  - 품사와 같은 문법적인 요소와 더 의미적인 부분을 감안하여 정확한 철자로 된 어근 단어를 찾아줌
  - 변환 시간이 오래 걸림

#8.2 텍스트 정규화

Stemming

```
from nltk.stem import LancasterStemmer
stemmer = LancasterStemmer()

print(stemmer.stem('working'),stemmer.stem('works'),stemmer.stem('worked'))
print(stemmer.stem('amusing'),stemmer.stem('amuses'),stemmer.stem('amused'))
print(stemmer.stem('happier'),stemmer.stem('happiest'))
print(stemmer.stem('fancier'),stemmer.stem('fanciest'))
```

```
work work work
amus amus amus
happy happiest
fant fanciest
```

#8.2 텍스트 정규화

Lemmatization

```
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet')

lemma = WordNetLemmatizer()
print(lemma.lemmatize('amusing','v'),lemma.lemmatize('amuses','v'),lemma.lemmatize('amused','v'))
print(lemma.lemmatize('happier','a'),lemma.lemmatize('happiest','a'))
print(lemma.lemmatize('fancier','a'),lemma.lemmatize('fanciest','a'))
```

```
amuse amuse amuse
happy happy
fancy fancy
```

8.3 Bag of Words - BOW

EURON 초급 세션 1팀 박혜진

EWHA EURON

개요

피처 벡터화/추출 : 피처를 추출하고 벡터 값을 할당하는 작업

대표적인 방법 : 1) BOW(Count 기반, TF-IDF 기반 벡터화)

2) Word2Vec

Bag of Words : **문맥/순서를 무시**하고, **일괄적으로 단어에 대해 빈도값을 부여해 피처 값을 추출**하는 모델

장점 : 쉽고 빠른 구축

단점 : 문맥 의미 반영 부족, 희소 행렬 문제

1) BOW 피처 벡터화

BOW 모델에서의 피처 벡터화 :

모든 문서에서 모든 단어를 칼럼 형태로 나열하고,

각 문서에서 해당 단어의 횟수나 정규화된 빈도를 값으로 부여하는 데이터 세트 모델로 변경하는 것

ex) M개의 텍스트 문서에서 추출한 모든 단어가 N개 => M X N 개의 단어 피처로 이루어진 행렬 생성

방식 : 1) **카운트 기반의 벡터화**

-> **count** : 각 문서에서 해당 단어가 나타나는 횟수

-> 카운트 값이 높을수록 중요한 단어

-> 문제점 : 언어 특성상 자주 사용되는 단어까지 높은 값이 부여됨 => 2)

2) **TF-IDF(Term Frequency - Inverse Document Frequency) 벡터화**

-> **개별 문서**에서 자주 나타나는 단어에는 **가중치**를,

모든 문서에서 자주 나타나는 단어에는 **페널티**를 부여

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

TF_i = 개별 문서에서의 단어 i 빈도

DF_i = 단어 i를 가지고 있는 문서 개수

N = 전체 문서 개수

2) 사이킷런의 Count 및 TF-IDF 벡터화 구현

사이킷런의 CountVectorizer 클래스 : **카운트 기반의 벡터화**를 구현한 클래스

-> 피처 벡터화와 더불어 **텍스트 전처리**도 함께 수행

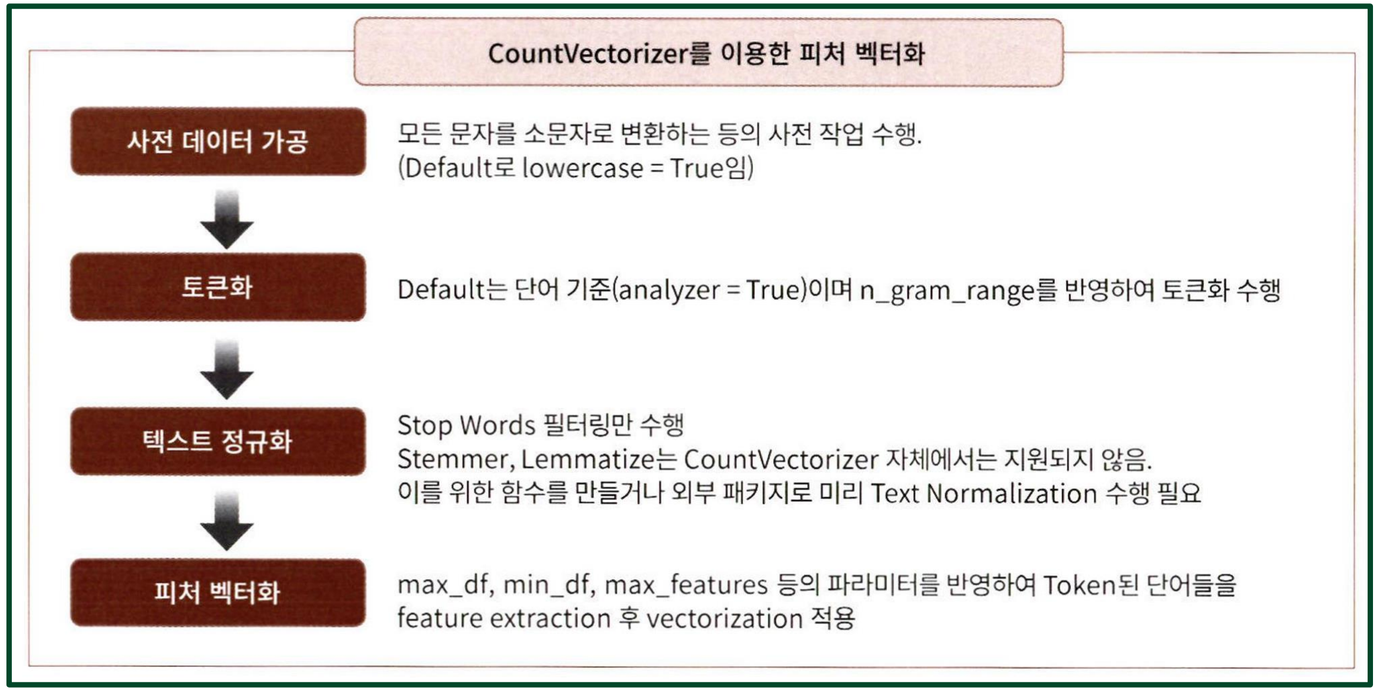
-> **fit(), transform()**을 통해 피처 벡터화된 객체 반환

-> 입력 파라미터

- ✓ **max_df** : 너무 높은 빈도수를 가지는 단어 피처 제외
- ✓ **min_df** : 너무 낮은 빈도수를 가지는 단어 피처 제외
- ✓ **max_features** : 추출하는 피처의 개수를 정수 값으로 지정해 제한
- ✓ **stop_words** : 추출에서 제외하는 단어 지정
- ✓ **n_gram_range** : 단어 순서를 보강하기 위해 n_gram 범위를 튜플 형태로 지정
- ✓ **analyzer** : 피처 추출을 수행한 단위
- ✓ **token_pattern** : 토큰화를 수행하는 정규 표현식 패턴 지정
- ✓ **tokenizer** : 토큰화를 별도의 커스텀 함수로 이용시 적용

2) 사이킷런의 Count 및 TF-IDF 벡터화 구현

-> **CountVectorizer를 이용한 피처 벡터화 프로세스**



TfidfVectorizer 클래스 : 사이킷런에서 TF-IDF 벡터화를 구현한 클래스

-> 파라미터와 변환 방법은 동일

3) BOW 벡터화를 위한 희소 행렬

BOW 형태를 가진 언어 모델의 피처 벡터화는 대부분 희소 행렬이다

-> 문제점 : 메모리 공간이 많이 필요

데이터 액세스를 위한 시간 소모가 커짐

=> 희소 행렬이 적은 메모리 공간을 차지하도록 변환해야 함

방법 : **COO 형식, CSR 형식**

수십만 개의 칼럼

	단어 1	단어 2	단어 3	....	단어 1000	....	단어 2000	....	단어 10000	....	단어 20000	....	단어 100000
수천 ~ 수만개 레코드	문서1	1	2	2	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

4) 희소 행렬 - COO 형식

COO(Coordinate) 형식 : **0이 아닌 데이터만** 별도의 데이터 배열에 저장하고,
데이터가 가리키는 **행과 열의 위치를 별도의 배열**로 저장

-> 파이썬에서는 희소 행렬 변환을 위해 주로 **Scipy** 사용 : **sparse 패키지**

[3, 1, 2], [0, 2, 0]

-> 0이 아닌 데이터 : [3, 1, 2]

-> (row, col)로 표시 : (0,0), (0,2), (1,1)

-> row, col을 별도의 배열로 저장 :

-> row : [0, 0, 1]

-> col : [0, 2, 1]

4) 희소 행렬 - COO 형식

1) 2차원 데이터를 넘파이의 ndarray 객체로 생성

```
import numpy as np

dense = np.array([3, 0, 1], [0, 2, 0])
```

2) 밀집 행렬을 사이파이의 **coo_matrix 클래스**로 COO 형식의 희소 행렬로 변환

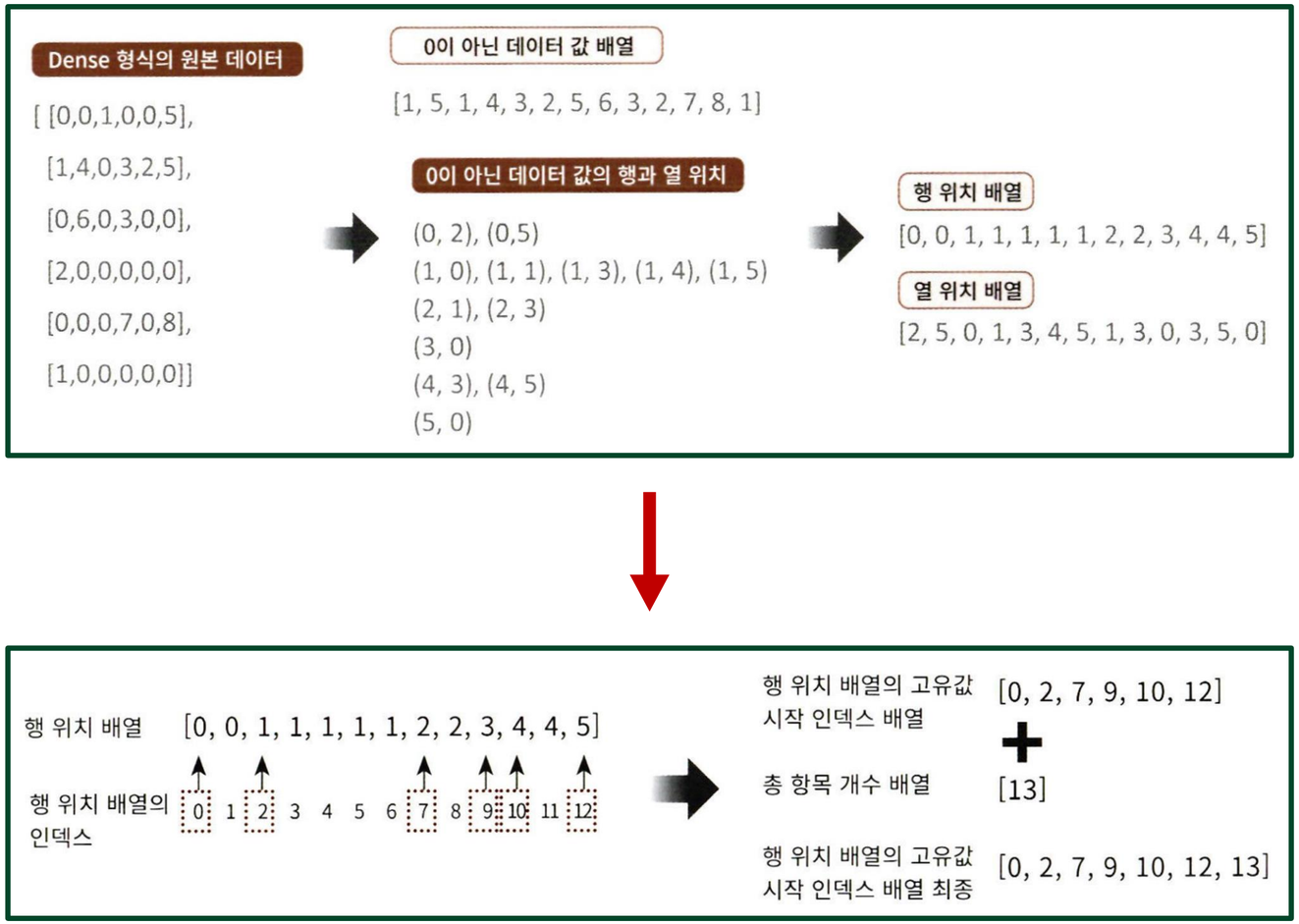
```
from scipy import sparse

# 0 이 아닌 데이터 추출해 별도의 배열 데이터로 만듬
data = np.array([3,1,2])

# 행 위치와 열 위치를 각각 배열로 생성
row_pos = np.array([0, 0, 1])
col_pos = np.array([0, 2, 1])

# 만든 행 위치와 열 위치 배열을 coo_matrix() 내에 생성 파라미터로 입력
sparse_coo = sparse.coo_matrix((data, (row_pos, col_pos)))
```

3) **tooarray() 메서드**로 다시 원래의 밀집 형태의 행렬로 출력

```
sparse_coo_tooaray()
```

5) 희소 행렬 - CSR 형식

CSR(Compressed Sparse Row) 형식 : COO 형식의 문제점(반복적인 위치 데이터 사용)을 해결한 방식



5) 희소 행렬 - CSR 형식

CSR 방식의 변환 : 사이파이의 **csr_matrix 클래스**

```
from scipy import sparse

dense2 = np.array([[0, 0, 1, 0, 0, 5],
                   [1, 4, 0, 3, 2, 5],
                   [0, 6, 0, 3, 0, 0],
                   [2, 0, 0, 0, 0, 0],
                   [0, 0, 0, 7, 0, 8],
                   [1, 0, 0, 0, 0, 0]])

# 0이 아닌 데이터 추출
data2 = np.array([1, 5, 1, 4, 3, 2, 5, 6, 3, 2, 7, 8, 1])

# 행 위치와 열 위치를 각각 array로 생성
row_pos = np.array([0, 0, 1, 1, 1, 1, 1, 2, 2, 3, 4, 4, 5])
col_pos = np.array([2, 5, 0, 1, 3, 4, 5, 1, 3, 0, 3, 5, 0])

# COO 형식으로 변환
sparse_coo = sparse.coo_matrix((data2, (row_pos, col_pos)))

# 행 위치 배열의 고유한 값의 시작 위치 인덱스를 배열로 생성
row_pos_ind = np.array([0, 2, 7, 9, 10, 12, 13])

# CSR 형식으로 반환
sparse_csr = sparse.csr_matrix((data2, col_pos, row_pos_ind))
```

```
COO 변환된 데이터가 제대로 되었는지 다시 Dense로 출력 확인
[[0 0 1 0 0 5]
 [1 4 0 3 2 5]
 [0 6 0 3 0 0]
 [2 0 0 0 0 0]
 [0 0 0 7 0 8]
 [1 0 0 0 0 0]]
CSR 변환된 데이터가 제대로 되었는지 다시 Dense로 출력 확인
[[0 0 1 0 0 5]
 [1 4 0 3 2 5]
 [0 6 0 3 0 0]
 [2 0 0 0 0 0]
 [0 0 0 7 0 8]
 [1 0 0 0 0 0]]
```

감성 분석

초급 1팀 권나영

EWHA EURON

#01. 감성 분석 소개

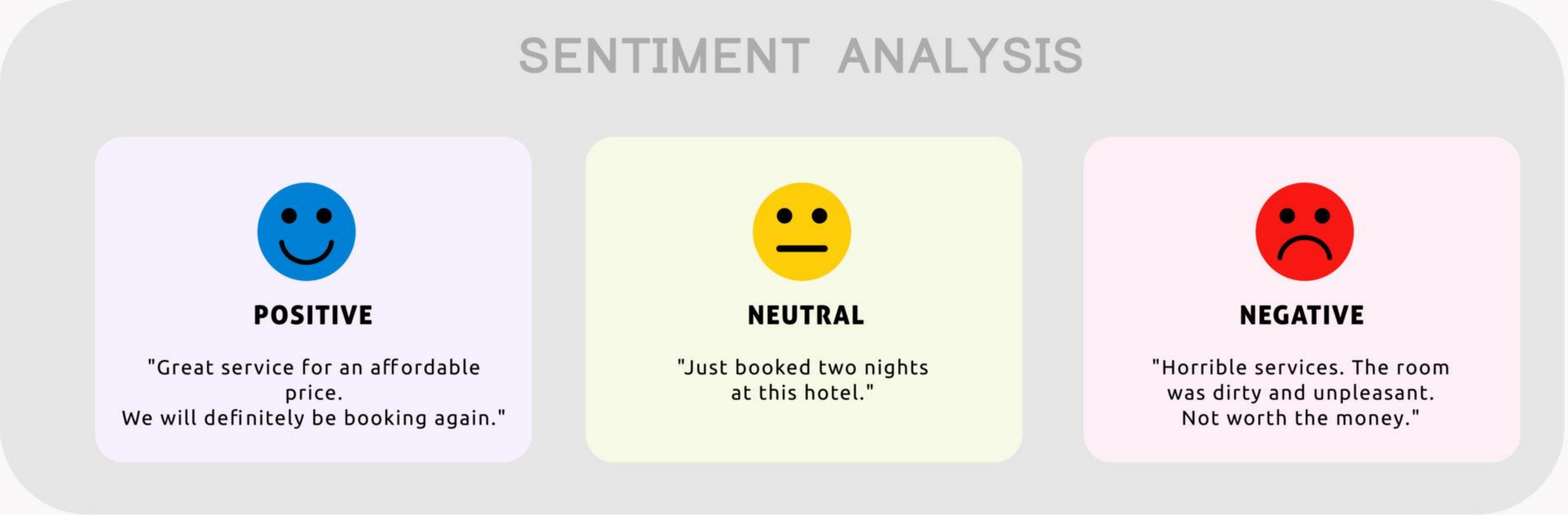
감성 분석?

문서의 주관적인 감성/의견/감정/기분 등을 파악하기 위한 방법
소셜 미디어, 여론조사, 온라인 리뷰, 피드백 등 다양한 분야에서 활용

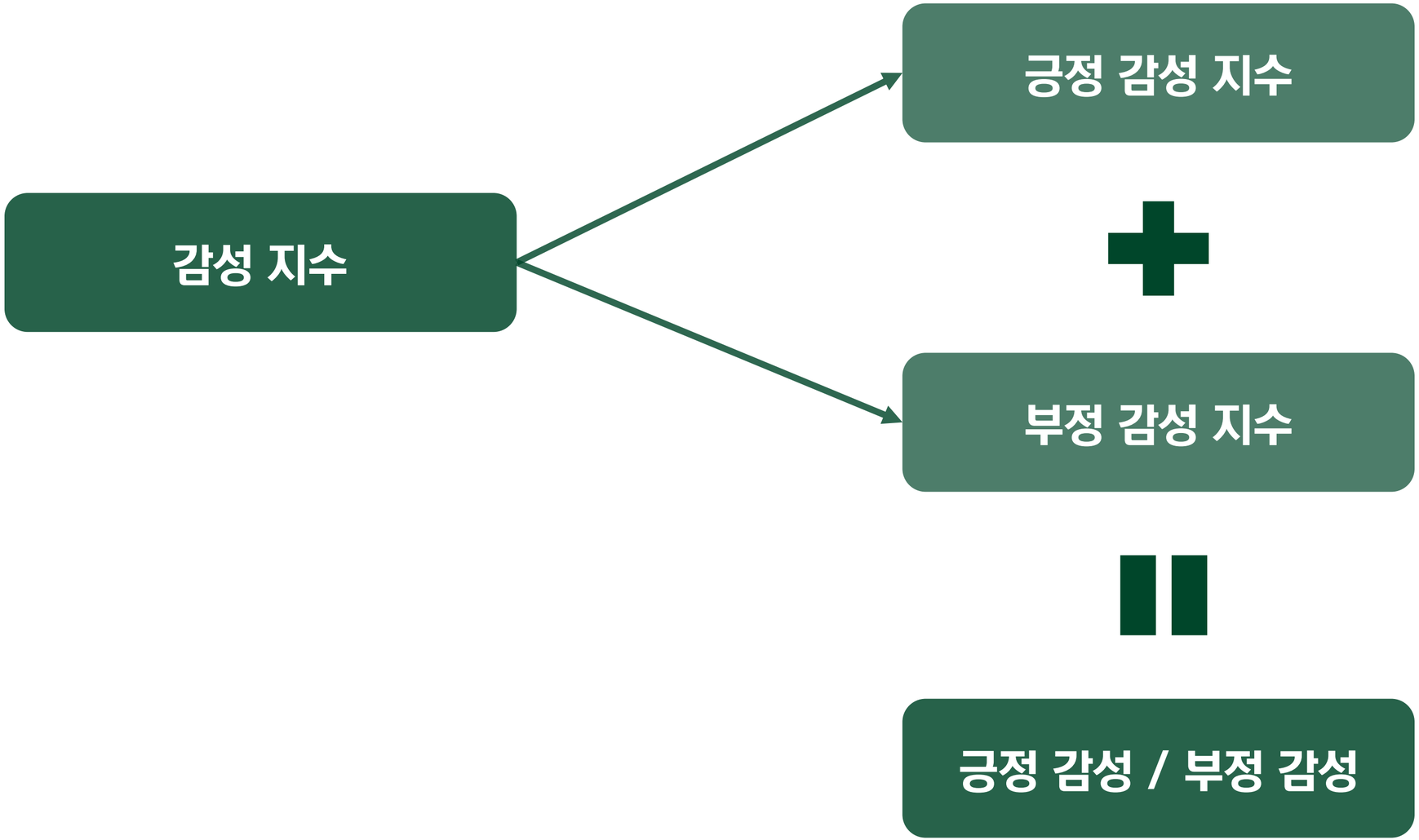
How?

문서 내 텍스트가 나타내는 여러 가지 주관적인 단어와 문맥을 기반으로 감성(Sentiment) 수치를 계산

#01. 감성 분석 소개


SENTIMENT ANALYSIS
POSITIVE
"Great service for an affordable price.
We will definitely be booking again."
NEUTRAL
"Just booked two nights at this hotel."
NEGATIVE
"Horrible services. The room was dirty and unpleasant.
Not worth the money."

#01. 감성 분석 소개



#01. 감성 분석 소개

지도학습 방식

- 학습 데이터와 타깃 레이블 값을 기반으로 감성 분석 학습을 수행한 뒤 이를 기반으로 다른 데이터의 감성 분석을 예측
- 일반적인 텍스트 기반의 분류와 거의 동일

비지도 학습 방식

- ‘Lexicon’ 이라는 일종의 감성 어휘 사전 이용

#02 지도학습 기반 감성 분석 실습 – IMDB 영화평

```
import pandas as pd

review_df = pd.read_csv('./labeledTrainData.tsv', header=0, sep="₩t", quoting=3)
review_df.head(3)
```

	id	sentiment	review
0	"5814_8"	1	"With all this stuff going down at the moment ...
1	"2381_9"	1	"\"The Classic War of the Worlds\" by Timothy ...
2	"7759_3"	0	"The film starts with a manager (Nicholas Bell...

labeledTrainData.tsv 파일: 탭 문자로 분리됨
➔ sep="\t" 명시를 통해 DataFrame으로 로딩

피처
- id: 각 데이터의 id
- sentiment: 영화평의 setiment 결과 값
  1은 긍정적 평가, 0은 부정적 평가
- review: 영화평의 텍스트

EWHA EURON

#02 지도학습 기반 감성 분석 실습 – IMDB 영화평

```
import re

# <br> html 태그는 replace 함수로 공백으로 변환
review_df['review'] = review_df['review'].str.replace('<br />',' ')

# 파이썬의 정규 표현식 모듈인 re를 이용하여 영어 문자열이 아닌 문자는 모두 공백으로 변환
review_df['review'] = review_df['review'].apply( lambda x : re.sub("[^a-zA-Z]", " ", x) )
```

파이썬 re 모듈

정규 표현식 지원
[^a-zA-z]: 영어 대/소문자가 아닌 모든 문자를 찾는 것

#02 지도학습 기반 감성 분석 실습 – IMDB 영화평

```
from sklearn.model_selection import train_test_split

class_df = review_df['sentiment']
feature_df = review_df.drop(['id','sentiment'], axis=1, inplace=False)

X_train, X_test, y_train, y_test= train_test_split(feature_df, class_df, test_size=0.3, random_state=156)

X_train.shape, X_test.shape
```

```
((17500, 1), (7500, 1))
```

학습용 데이터: 17500개의 리뷰
테스트용 데이터: 7500개의 리뷰

#02 지도학습 기반 감성 분석 실습 – IMDB 영화평

Count 벡터화를 적용해 예측 성능 측정

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score

# 스톱 워드는 English, filtering, ngram은 (1,2)로 설정해 CountVectorization수행.
# LogisticRegression의 C는 10으로 설정.
pipeline = Pipeline([
    ('cnt_vect', CountVectorizer(stop_words='english', ngram_range=(1,2) )),
    ('lr_clf', LogisticRegression(C=10))])

# Pipeline 객체를 이용하여 fit(), predict()로 학습/예측 수행. predict_proba()는 roc_auc때문에 수행.
pipeline.fit(X_train['review'], y_train)
pred = pipeline.predict(X_test['review'])
pred_probs = pipeline.predict_proba(X_test['review'])[:,1]

print('예측 정확도는 {0:.4f}, ROC-AUC는 {1:.4f}'.format(accuracy_score(y_test ,pred),
                                            roc_auc_score(y_test, pred_probs)))
```

```
C:₩Users₩82102₩anaconda3₩lib₩site-packages₩sklearn₩linear_model₩_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
예측 정확도는 0.8860, ROC-AUC는 0.9503
```

EWHA EURON

#02 지도학습 기반 감성 분석 실습 – IMDB 영화평

TF-IDF 벡터화를 적용해 예측 성능 측정

```
# 스톱 워드는 english, filtering, ngram은 (1,2)로 설정해 TF-IDF 벡터화 수행.
# LogisticRegression의 C는 10으로 설정.
pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(stop_words='english', ngram_range=(1,2) )),
    ('lr_clf', LogisticRegression(C=10))])

pipeline.fit(X_train['review'], y_train)
pred = pipeline.predict(X_test['review'])
pred_probs = pipeline.predict_proba(X_test['review'])[:,1]

print('예측 정확도는 {0:.4f}, ROC-AUC는 {1:.4f}'.format(accuracy_score(y_test ,pred),
                                                roc_auc_score(y_test, pred_probs)))
```

```
예측 정확도는 0.8936, ROC-AUC는 0.9598
```

#03 비지도학습 기반 감성 분석 소개

Lexicon

- 비지도 감성 분석의 기반
- 감성 어휘 사전
- 감성 지수: 긍정 감성 또는 부정 감성의 정도를 의미하는 수치를 가지고 있음
- 대표 패키지: NLTK 패키지

WorkNet

- NLP에서 제공하는 방대한 영어 어휘 사전
- 시맨틱 분석을 제공. 즉, 문맥상 의미를 포함하는 정보 제공

#03 비지도학습 기반 감성 분석 소개

대표적인 NLTK 감성 사전

NLTK의 예측 성능이 좋지 않기 때문에 실제 업무에는 NLTK 패키지가 아닌 다른 감성 사전 적용

SentiWordNet
- WordNet의 Synset 개념을 감성 분석에 적용하여 3가지 감성 점수 할당
- 긍정 감성 지수, 부정 감성 지수, 객관성 지수
- 예측 정확도가 높지 않음

VADER
- 소셜 미디어의 텍스트에 대한 감성 분석 제공 패키지
- 뛰어난 감성 분석 결과 제공 및 비교적 빠른 수행 시간 -> 대용량 텍스트 데이터에 사용

Pattern

예측 성능 측면에서 가장 주목

#04 SentiWordNet을 이용한 감성 분석

```
from nltk.corpus import wordnet as wn

term = 'present'

# 'present'라는 단어로 wordnet의 synsets 생성.
synsets = wn.synsets(term)
print('synsets() 반환 type :', type(synsets))
print('synsets() 반환 값 갯수:', len(synsets))
print('synsets() 반환 값 :', synsets)
```

```
synsets() 반환 type : <class 'list'>
synsets() 반환 값 갯수: 18
synsets() 반환 값 : [Synset('present.n.01'), Synset('present.n.02'), Synset('present.n.03'), Synset('show.v.01'), Synset('present.v.02'), Synset('stage.v.01'), Synset('present.v.04'), Synset('present.v.05'), Synset('award.v.01'), Synset('give.v.08'), Synset('deliver.v.01'), Synset('introduce.v.01'), Synset('portray.v.04'), Synset('confront.v.03'), Synset('present.v.12'), Synset('salute.v.06'), Synset('present.a.01'), Synset('present.a.02')]
```

POS 태그

present.n.01

- present: 의미
- n: 명사 품사
- 01: present가 명사로서 가지는 의미를 구분하는 인덱스

Present 단어에 대한 Synset 추출

WordNet의 synsets( ) :

- 파라미터로 지정된 단어에 대해 WordNet에 등재된 모든 Synset 객체 반환
- 여러 개의 Synset 객체를 가지는 리스트 반환
- Synset 객체의 파라미터는 POS 태그를 나타냄

EWHA EURON

#04 SentiWordNet을 이용한 감성 분석

```
for synset in synsets :
    print('##### Synset name : ', synset.name(),'#####')
    print('POS :',synset.lexname())
    print('Definition:',synset.definition())
    print('Lemmas:',synset.lemma_names())
```

```
##### 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']
##### Synset name :  show.v.01 #####
POS : verb.perception
Definition: give an exhibition of to an interested audience
Lemmas: ['show', 'demo', 'exhibit', 'present', 'demonstrate']
##### Synset name :  present.v.02 #####
POS : verb.communication
Definition: bring forward and present to the mind
Lemmas: ['present', 'represent', 'lay_out']
```

명사지만 서로 다른 의미를 가진다

#04 SentiWordNet을 이용한 감성 분석

```
# synset 객체를 단어별로 생성합니다.
tree = wn.synset('tree.n.01')
lion = wn.synset('lion.n.01')
tiger = wn.synset('tiger.n.02')
cat = wn.synset('cat.n.01')
dog = wn.synset('dog.n.01')

entities = [tree , lion , tiger , cat , dog]
similarities = []
entity_names = [ entity.name().split('.')[0] for entity in entities]

# 단어별 synset 들을 iteration 하면서 다른 단어들의 synset과 유사도를 측정합니다.
for entity in entities:
    similarity = [ round(entity.path_similarity(compared_entity), 2)  for compared_entity in entities ]
    similarities.append(similarity)

# 개별 단어별 synset과 다른 단어의 synset과의 유사도를 DataFrame형태로 저장합니다.
similarity_df = pd.DataFrame(similarities , columns=entity_names,index=entity_names)
similarity_df
```

path_similarity( ) 메서드를 통해 단어 간 상호 유사도 측정

	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
triger과는 유사도가 0.33으로 가장 크다

EWHA EURON

#04 SentiWordNet을 이용한 감성 분석

```
import nltk
from nltk.corpus import sentiwordnet as swn

father = swn.senti_synset('father.n.01')
print('father 긍정감성 지수: ', father.pos_score())
print('father 부정감성 지수: ', father.neg_score())
print('father 객관성 지수: ', father.obj_score())
print('₩n')
fabulous = swn.senti_synset('fabulous.a.01')
print('fabulous 긍정감성 지수: ',fabulous .pos_score())
print('fabulous 부정감성 지수: ',fabulous .neg_score())
```

```
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
```

#04 SentiWordNet을 이용한 감성 분석

SentiWordNet을 이용한 영화 감상평 분석

1. 문서를 문장 단위로 분해
2. 다시 문장을 단어 단위로 토큰화하고 품사 태깅
3. 품사 태깅된 단어 기반으로 synset 객체와 senti_synset 객체 생성
4. Senti_synset에서 긍정 감성/부정 감성 지수를 구하고 이를 모두 합산해 특정 임계치 값 이상일 때 긍정 감성으로 그렇지 않을 때는 부정 감성으로 결정

#04 SentiWordNet을 이용한 감성 분석

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
from sklearn.metrics import recall_score, f1_score, roc_auc_score

print(confusion_matrix( y_target, preds))
print("정확도:", accuracy_score(y_target , preds))
print("정밀도:", precision_score(y_target , preds))
print("재현율:", recall_score(y_target, preds))
```

```
[[7668 4832]
 [3636 8864]]
정확도: 0.66128
정밀도: 0.6471962616822243
재현율: 0.70912
```

#05 VADER를 이용한 감성 분석

VADER Lexicon

- 소셜 미디어 감성 분석 용도로 만들어진 룰 기반의 Lexicon
- SentimentIntensityAnalyzer 클래스를 이용해 감성 분석 제공
- NLTK의 서브 모듈 뿐만 아니라 단독 패키지로도 제공됨

#05 VADER를 이용한 감성 분석

VADER 감성 분석 수행 과정

1. SentimentIntensityAnalyzer 객체 생성
2. 문서별로 polarity_scores( ) 메서드 호출 통해 감성 점수 계산: 딕셔너리 형태로 반환
3. 계산한 감성 점수가 특정 임계값 이상이면 긍정, 그렇지 않으면 부정으로 판단

'neg' : 부정 감성 지수, 'neu' : 중립 감성 지수, 'pos' : 긍정 감성 지수,
compound: 부정/중립/긍정 지수를 적절히 조합하여 -1~1 사이의 지수를 표현한 것

Compound score을 기반으로 부정 감성 또는 긍정 감성 여부 결정
보통 0.1 이상이면 긍정 감성으로 판단

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

senti_analyzer = SentimentIntensityAnalyzer()
senti_scores = senti_analyzer.polarity_scores(review_df['review'][0])
print(senti_scores)
```

```
{'neg': 0.13, 'neu': 0.743, 'pos': 0.127, 'compound': -0.7943}
```

#05 VADER를 이용한 감성 분석

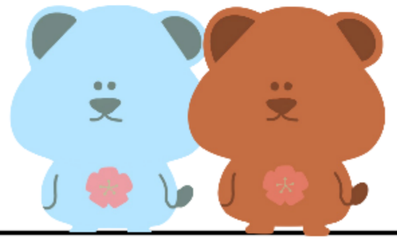
```
def vader_polarity(review,threshold=0.1):
    analyzer = SentimentIntensityAnalyzer()
    scores = analyzer.polarity_scores(review)

    # compound 값에 기반하여 threshold 입력값보다 크면 1, 그렇지 않으면 0을 반환
    agg_score = scores['compound']
    final_sentiment = 1 if agg_score >= threshold else 0
    return final_sentiment

# apply lambda 식을 이용하여 레코드별로 vader_polarity( )를 수행하고 결과를 'vader_preds'에 저장
review_df['vader_preds'] = review_df['review'].apply( lambda x : vader_polarity(x, 0.1) )
y_target = review_df['sentiment'].values
vader_preds = review_df['vader_preds'].values
```

```
print('#### VADER 예측 성능 평가 ####')
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
from sklearn.metrics import recall_score, f1_score, roc_auc_score

print(confusion_matrix( y_target, vader_preds))
print("정확도:", accuracy_score(y_target , vader_preds))
print("정밀도:", precision_score(y_target , vader_preds))
print("재현율:", recall_score(y_target, vader_preds))
```

```
#### VADER 예측 성능 평가 ####
[[ 6736  5764]
 [ 1867 10633]]
정확도: 0.69476
정밀도: 0.6484722815149113
재현율: 0.85064
```

EWHA EURON

THANK YOU



EWHA EURON