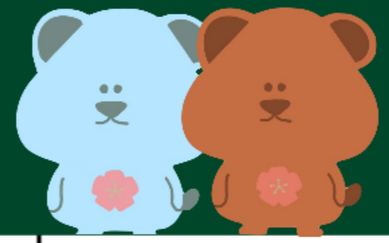
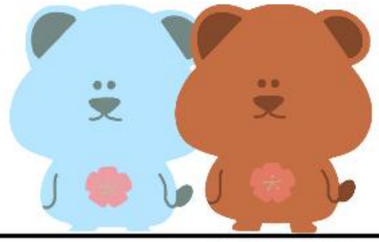


8장 텍스트 분석

2팀 강민서, 박나림, 이가은

EWHA EURON

목차



EWHA EURON

8.0 텍스트 분석 개요

EWHA EURON

#0 텍스트 분석 개요

#1 텍스트 분석이란?

- **비정형 텍스트 데이터에서 의미 있는 정보 추출**
- **머신러닝·언어 이해 기술을 활용**
- 텍스트를 숫자 기반 피처로 변환 필요
→ 피처 벡터화 또는 피처 추출
- NLP **기술 발전 → 더 정교한 분석 가능**

#2 NLP vs 텍스트 분석

구분	NLP	텍스트 분석(TA)
초점	언어 이해 및 처리 기술	의미 추출 및 활용
예시	번역, 질의 응답 시스템	여론 분석, 정책 전략, 비즈니스 정보 추출

☞ NLP는 *언어 처리 기술 자체*
텍스트 분석은 *텍스트를 사용해 정보를 얻고 예측/분석*
(둘은 상호 보완 관계)

#3 활용 분야

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	

#4 주요 기술 영역

1. 텍스트 분류
: 문서를 특정 카테고리로 분류
예) 스팸 탐지, 뉴스 분류
→ 지도학습 기반

2. 감성 분석
: 텍스트에 담긴 감정·의견 분석
예) SNS 긍/부정, 리뷰 분석
→ 지도/비지도 모두 활용

3. 텍스트 요약
: 핵심 주제·내용 자동 추출
예) 토픽 모델링 활용

4. 텍스트 군집화 &유사도 분석
: 비슷한 문서 묶기 (비지도학습)
예) 문서 추천, 탐색에 활용

EWHA EURON

8.1 텍스트 분석 이해

EWHA EURON

#1 텍스트 분석 이해

#1-1 텍스트 분석 주요 Task

Task	설명	대표 기술
전처리	토크나이징, stopwords 제거, 정규화	nltk, spaCy, KoNLPy
벡터화	텍스트 → 숫자 벡터	BOW, TF-IDF, Word2Vec
감성 분석	의견/감정 분석	ML, LSTM, transformers
분류	카테고리 자동 분류	SVM, Naive Bayes
군집화	유사 문서 묶음	K-means, LDA

#1-2 텍스트 분석 프로세스

1. **텍스트 사전 처리**
클렌징, 소문자 변환, 불필요한 용어 제거, 토큰화, 어간/표제어 처리

2. 피처 벡터화
BOW(단어 발생 횟수 기반)
TF-IDF(중요도 반영)
Word2Vec(의미적 유사성 벡터)

3. 모델 학습 & 예측
분류, 감성 분석, 주제 모델링 등
➡ 텍스트 → 피처 → ML 모델 → 결과 해석 흐름

#2 파이썬 텍스트 분석 패키지

#2-1 NLTK
- 파이썬의 가장 오래된 NLP 패키지
- 자연어 처리 & 문서 분석용 파이썬 라이브러리
  → 다만 성능이 느리고 대규모 처리엔 부적합

* 주요 기능
- 토큰화
- 품사 태깅
- 형태소 분석
- 말뭉치 제공

#2-2 Gensim
- 파이썬에서 제공하는 Word2Vec 라이브러리
- 대규모 텍스트 처리에 효율적(스트리밍 처리)

* 주요 기능
- 임베딩 : Word2Vec
- 토픽 모델링
- LDA

#2-3 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()

#2-4 SpaCy
- 최신·고성능 NLP 패키지
- 속도 빠르고 산업환경 적용에 적합
- 품사 태깅, 의존 구문 분석, NER 강점

* 주요 기능
- 고성능 토큰화
- NER(Named Entity Recognition)
- 의존 구문 분석
- 벡터 기반 의미 처리

8.2 텍스트 정규화

EWHA EURON

#1 텍스트 정규화

텍스트 정규화

텍스트를 머신러닝 알고리즘이나 NLP 애플리케이션에
입력 데이터로 사용하기 위해
다양한 텍스트 데이터의 사전 작업을 수행하는 것

클렌징 (Cleansing)
토큰화 (Tokenization)
필터링 / 스톱 워드 제거 / 철자 수정
Stemming
Lemmatization

#1 클렌징 & 텍스트 토큰화

클렌징

텍스트에서 분석에 방해되는 불필요한 문자, 기호 등을 사전에 제거하는 작업
ex) HTML, XML 태그나 특정 기호 등을 사전에 제거

텍스트 토큰화

문서에서 문장을 분리하는 문장 토큰화, 문장에서 단어를 토큰으로 분리하는 단어 토큰화로 나눌 수 있다.

#1 텍스트 토큰화

문장 토큰화(sent_tokenize)

- 문장의 마침표(.), 개행문자(\n) 등 문장의 마지막을 뜻하는 기호에 따라 분리
- 정규 표현식에 따른 문장 토큰화도 가능

```
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 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 your window or your television.', 'You feel it when you go to work, or go to church or pay your taxes.']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

#1 텍스트 토큰화

단어 토큰화(word_tokenize)

- 문장을 단어로 토큰화하는 것
- 기본적으로 공백, 콤마(,), 마침표(.), 개행문자(\n) 등으로 단어 분리
- 정규 표현식을 이용해 다양한 유형으로 토큰화 수행 가능

```
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', '.']
```

#1 텍스트 토큰화

단어 토큰화와 문서 토큰화 조합

```
from nltk import word_tokenize, sent_tokenize

#여러 개의 문장으로 된 입력 데이터를 문장별로 단어 토큰화하게 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 문장에 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', 'your', 'television', '.'],

['You', 'feel', 'it', 'when', 'you', 'go', 'to', 'work', ',', 'or', 'go', 'to', 'church', 'or', 'pay', 'your', 'taxes', '.']]
```

#1 텍스트 토큰화

단어 토큰화와 문서 토큰화 조합

문장을 단어별로 하나씩 토큰화 할 경우 문맥적 의미가 무시될 수 있음

➡ **n_gram**

- 연속된 n개의 단어를 하나의 토큰화 단위로 분리
- n개 단어 크기 윈도우를 만들어 문장의 처음부터 오른쪽으로 움직이면서 토큰화 수행

ex) 2-gram(bigram)

연속적으로 2개의 단어들을 순차적으로 이동하면서 단어들을 토큰화
- “Agent Smith knocks the door”
- (Agent, Smith), (Smith, knocks), (knocks, the), (the, door)

#2 스톱 워드 제거

스톱 워드

- 분석에 큰 의미가 없는 단어를 지칭
  ex) 영어에서 is, the, a, will 등 문맥적으로 큰 의미가 없는 단어

- 사전에 제거하지 않으면 빈번함으로 인해 오히려 중요한 단어로 인지되므로 제거해야 하는 것이 중요한 전처리 작업

NLTK의 스톱워드

NLTK의 stopwords 목록 다운로드

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Stopwords 수 및 20개 확인

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

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

#2 스톱 워드 제거

스톱워드를 필터링으로 제거

```
import nltk

stopwords = nltk.corpus.stopwords.words('english')
all_tokens = []
#위 예제에서 3개의 문장별로 얻은 word_tokens list에 대해 스톱 워드를 제거하는 반복문
for sentence in word_tokens:
  filtered_words=[]
  #개별 문장별로 토큰화된 문장 list에 대해 스톱 워드를 제거하는 반복문
  for word in sentence:
    #소문자로 모두 변환합니다.
    word = word.lower()
    #토큰화된 개별 단어가 스톱 워드의 단어에 포함되지 않으면 word_tokens에 추가
    if word not in stopwords:
      filtered_words.append(word)
  all_tokens.append(filtered_words)

print(all_tokens)
```

```
[['matrix', 'everywhere', 'around', 'us', ',', 'even', 'room', '.'], ['see', 'window', 'television', '.'],

['feel', 'go', 'work', ',', 'go', 'church', 'pay', 'taxes', '.']]
```

#3 Stemming과 Lemmatization

많은 언어에서 문법적인 요소에 따라 단어가 다양하게 변화
Stemming과 Lemmatization은 문법적 또는 의미적으로 변화하는 단어의 원형을 찾는 것

Stemming

- 원형 단어로 변환 시 일반적인 방법을 적용하거나, 더 단순화된 방법을 적용
- 일부 철자가 훼손된 어근 단어를 추출하는 경향이 있음

Lemmatization

- 더 정교하며, 의미론적인 기반에서 단어의 원형을 찾음
- 품사와 같은 문법적인 요소와 더 의미적인 부분을 감안해 정확한 철자로 된 어근 단어를 찾아줌
- 변환 시간이 더 오래 걸림

#3 Stemming과 Lemmatization

NLTK에서 Stemming (LancasterStemming)

```
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
```

amus를 원형 단어로 인식

원형 단어에서 철자가 다른 어근 단어로 인식

#3 Stemming과 Lemmatization

NLTK에서 Lemmatization (WordNetLemmatizer)

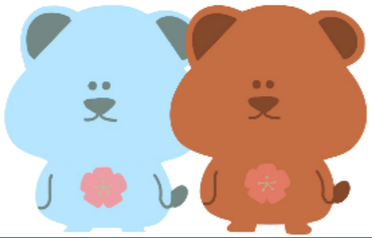
- 보다 정확한 원형 단어 추출을 위해 단어의 '품사 ' 를 입력해야 함
  ex) 동사 'v', 형용사 'a'
- Stemming보다 정확하게 원형 단어를 추출해줌

```
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'))
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
amuse amuse amuse
happy happy
fancy fancy
```

3. Bag of Words-BOW



EWHA EURON

#01 BOW란?

#1 Bag of Words

: 문맥이나 순서를 무시하고 일괄적으로 단어에 대해 빈도 값을 부여해 피처 값을 추출하는 모델

1. 문장에 있는 모든 단어에서 중복을 제거하고 각 단어를 칼럼 형태로 나열
2. 각 단어에 고유의 인덱스를 부여
3. 해당 단어가 문장에서 나타나는 횟수를 각 단어 인덱스에 기재

#2 장단점

장점	1. 쉽고 빠른 구축 2. 문서의 특징을 잘 나타내어 여러 분야에 활용 가능
단점	1. 문맥 의미 반영 부족 -> 단어의 순서를 고려하지 않아 문장 내 의미적 관계나 맥락을 반영하지 못함. -> n-gram을 통해 일부 보완 가능하지만, 여전히 문맥적 해석이 어려움. 2. 희소 행렬 문제 -> 많은 문서에서 단어를 추출하다 보면 대부분의 단어가 특정 문서에는 등장하지 않아 0으로 채워진 거대한 희소 행렬이 만들어짐. -> 이러한 희소 행렬은 메모리 비효율 및 연산 속도 저하를 초래함.

#02 BOW 피처 벡터화

#1 피처 벡터화

- 텍스트를 벡터 값으로 변환하는 것
- 넓은 범위의 피처 추출에 포함
- 카운트 기반의 벡터화 / TF-IDF

#2 카운트 벡터화

- 단어 피처에 값을 부여할 때 각 문서에서 해당 단어가 나타나는 횟수를 부여
- 카운트 값이 높을수록 중요

=> 언어의 특성상 문장에서 자주 사용될 수 밖에 없는 단어까지 높은 값을 부여하게 된다.

#3 TF-IDF (Term Frequency-Inverse Document Frequency) 벡터화

- 자주 나타나는 단어에 높은 가중치
- 모든 문서에서 자주 나타나는 단어에 대해서는 페널티

=> 문서가 길고 개수가 많은 경우 TF-IDF 방식이 더 좋은 예측 성능을 보장

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

TF_i = 개별 문서에서의 단어 i 빈도

DF_i = 단어 i를 가지고 있는 문서 개수

N = 전체 문서 개수

#03 사이킷런의 Count 및 TF-IDF 벡터화 구현

#1 CountVectorizer 클래스

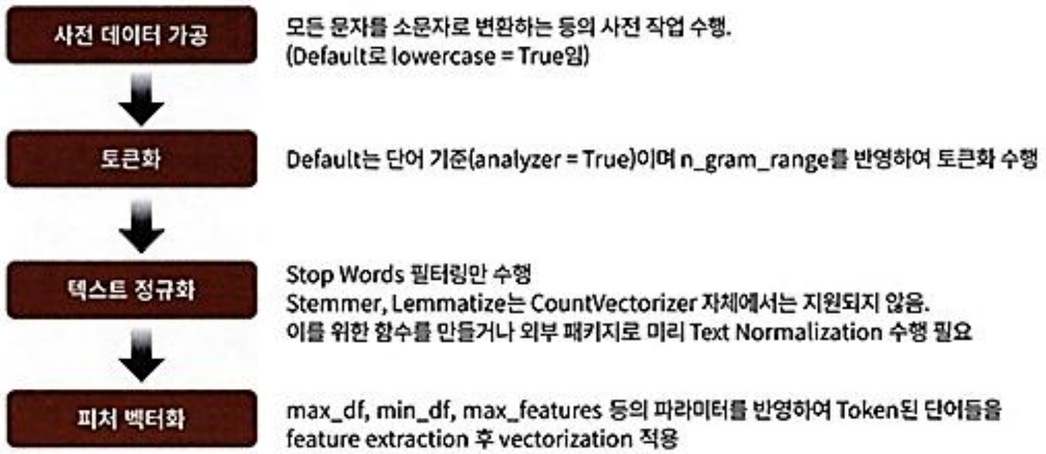
- 카운트 기반 벡터화를 수행하는 사이킷런의 클래스
- 소문자 일괄 변환, 토큰화, 스톱 워드 필터링 등 텍스트 전처리도 함께 수행
- fit(), transform()을 통해 피처 벡터화된 객체를 반환

#2 입력 파라미터

max_df	너무 자주 등장하는 단어를 제외하기 위한 기준 > 정수값인 경우: n개 이하로 나타나는 단어만 피처로 추출 > 소수값인 경우: 빈도수 하위 n% 단어만 피처로 추출
min_df	너무 드물게 등장하는 단어를 제외하기 위한 기준
max_features	추출할 피처 개수 제한
stop_words	해당 언어에서 스톱 워드로 지정된 단어 제거
ngram_range	튜플 형태로 n_gram 범위 설정 (단어를 몇 개씩 묶어 볼지 범위 지정)
analyzer	피처 추출을 수행할 단위 지정 > default: 'word'
token_pattern	토큰 정의를 위한 정규표현식 지정 > default: '\b\w\w+\b' (단어 분리가 사이의 2문자 이상의 단어를 토큰으로 분리)

#03 사이킷런의 Count 및 TF-IDF 벡터화 구현

#3 CountVectorizer 클래스를 이용한 피처 벡터화



#4 TfidVectorizer 클래스

- TF-IDF 벡터화를 수행하는 사이킷런의 클래스
- 파라미터와 변환 방법은 CountVectorizer와 동일

#04 BOW 벡터화를 위한 희소 행렬

#1 희소 행렬

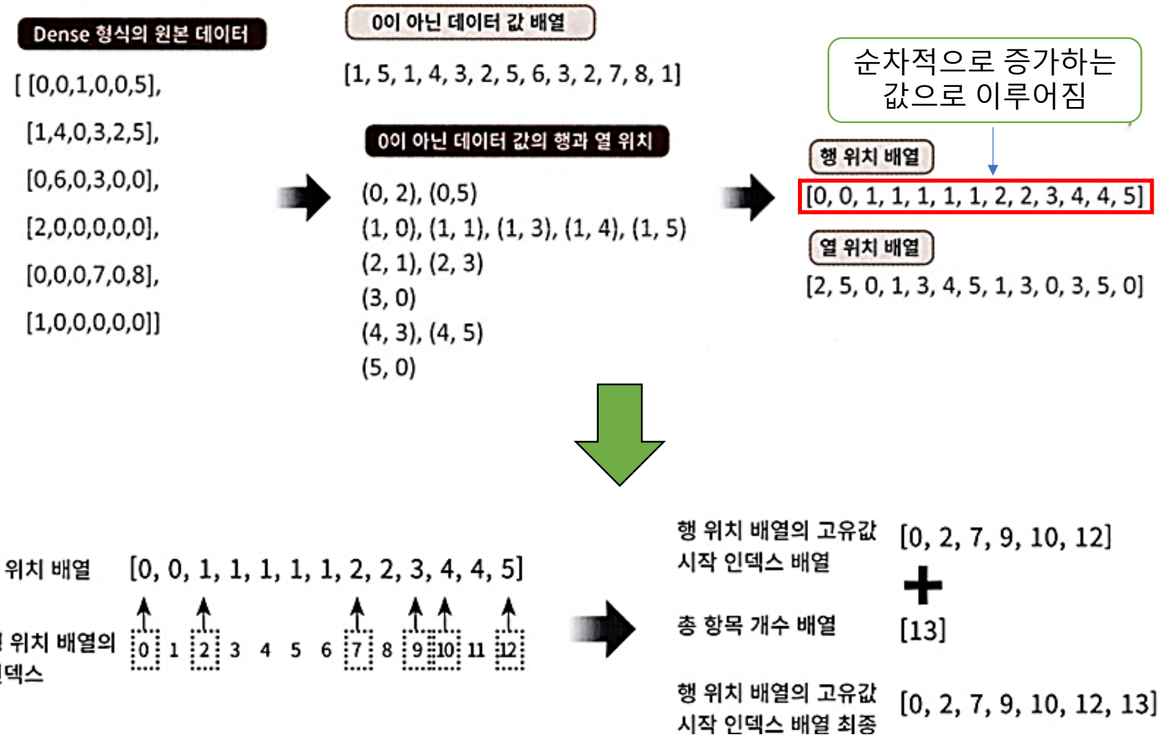
- **대부분의 값이 0**인 대규모 행렬
- BOW 기반의 텍스트 벡터화 결과는 대부분 희소 행렬 형태로 표현
- 메모리 공간이 많이 필요, 연산 시 데이터 액세스를 위한 시간이 많이 소모

=> COO형식 / CSR 형식으로 변환

#2 COO 형식

- 0이 아닌 데이터값 + 그 데이터가 저장된 행과 열의 위치를 별도로 저장

=> 문제점: 위치 데이터를 반복적으로 저장해서 데이터 중복이 많음



#3 CSR 형식

- 행 위치 배열의 고유한 값의 시작 위치만 표기
- 마지막에 데이터의 총 항목 개수 추가
- COO 방식모다 효율적

=> CountVectorizer/TfidfVectorizer는 CSR 형식의 희소 행렬 반환

#04 BOW 벡터화를 위한 희소 행렬

#4 sparse.coo_matrix

```python
import numpy as np
# 밀집행렬
dense=np.array([[3,0,1],[0,2,0]])

from scipy import sparse

# 0이 아닌 데이터 추출
data=np.array([3,1,2])

# 행 위치와 열 위치 저장 배열 생성
row_pos=np.array([0,0,1])
col_pos=np.array([0,2,1])

# COO 형식 희소 행렬 생성
sparse_coo=sparse.coo_matrix((data,(row_pos,col_pos)))

# 다시 밀집 형태로 출력
sparse_coo.toarray()
```

```
array([[3, 0, 1],
       [0, 2, 0]])
```

#5 sparse.csr_matrix

```python
from scipy import sparse
import numpy as np

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

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

print('COO 변환된 데이터가 제대로 되었는지 다시 Dense로 출력 확인')
print(sparse_coo.toarray())
```

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

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

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

print('CSR 변환된 데이터가 제대로 되었는지 다시 Dense로 출력 확인')
print(sparse_csr.toarray())
```

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

실제 사용 시에는 밀집 행렬을 생성 파라미터로 입력하면 COO나 CSR로 자동 생성 된다.

EWHA EURON

8.5 감성 분석

EWHA EURON

#1 감성 분석 소개

#1-1 감성 분석 개념

- 목적

: 문서에 담긴 주관적인 감성/의견/기분 등을 파악

- 활용 분야

: 소셜 미디어 글 / 여론조사 응답 / 온라인 리뷰 등

- 핵심 아이디어

: 문서 속의 주관적인 단어와 문맥을 기반으로 감성 점수를 계산

→ 보통 긍정 감성 지수와 부정 감성 지수,
두 지수를 계산한 뒤 이를 종합해 문서가
긍정적/부정적인지 판단

#1-2 머신러닝 관점에서의 감성 분석 방법

1-2-1. **지도학습 기반 감성 분석**

- **학습 데이터와 그에 대응하는 타깃 레이블을 이용**
- **이 데이터로 감성 분류 모델을 학습한 뒤, 새로운 데이터의 감성을 예측**

1-2-2. **비지도학습 기반 감성 분석**

- Lexicon(**감성 어휘 사전**) **사용**
- **문서에 포함된 단어들을** Lexicon**과 매칭하여 그 문서가 긍정인지 부정인지를 판단**
- **레이블이 붙은 학습 데이터 없이도 사용 가능**
- **‘단어 사전’의 품질에 따라 성능이 크게 좌우됨**

#2 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...

```
print(review_df['review'][0])
```

```
"With all this stuff going down at the moment with MJ i've started listening to his music, watching the
odd documentary here and there, watched The Wiz and watched Moonwalker again. Maybe i just want to get
a certain insight into this guy who i thought was really cool in the eighties just to maybe make up my
mind whether he is guilty or innocent. Moonwalker is part biography, part feature film which i remember
going to see at the cinema when it was originally released. Some of it has subtle messages about MJ's f
eeling towards the press and also the obvious message of drugs are bad m'kay.<br /><br />Visually impre
ssive but of course this is all about Michael Jackson so unless you remotely like MJ in anyway then you
are going to hate this and find it boring. Some may call MJ an egotist for consenting to the making of
this movie BUT MJ and most of his fans would say that he made it for the fans which if true is really n
ice of him.<br /><br />The actual feature film bit when it finally starts is only on for 20 minutes or
so excluding the Smooth Criminal sequence and Joe Pesci is convincing as a psychopathic all powerful dr
ug lord. Why he wants MJ dead so bad is beyond me. Because MJ overheard his plans? Nah, Joe Pesci's cha
racter ranted that he wanted people to know it is he who is supplying drugs etc so i dunno, maybe he ju
st hates MJ's music.<br /><br />Lots of cool things in this like MJ turning into a car and a robot and
the whole Speed Demon sequence. Also, the director must have had the patience of a saint when it came t
o filming the kiddy Bad sequence as usually directors hate working with one kid let alone a whole bunch
of them performing a complex dance scene.<br /><br />Bottom line, this movie is for people who like MJ
on one level or another (which i think is most people). If not, then stay away. It does try and give of
f a wholesome message and ironically MJ's bestest buddy in this movie is a girl! Michael Jackson is tru
ly one of the most talented people ever to grace this planet but is he guilty? Well, with all the atten
tion i've gave this subject....hmmm well i don't know because people can be different behind closed doo
rs, i know this for a fact. He is either an extremely nice but stupid guy or one of the most sickest li
ars. I hope he is not the latter."
```

#2-1

- **파일 형식**
  - `labeledTrainData.tsv`
  - **구분자: 탭 문자(`\t`)**
- **Pandas로 로딩**

: `sep="\t"` 인자를 주어 탭 구분 파일을 DataFrame으로 읽어들일 수 있음.

- **주요 칼럼**
  - `id` : **각 데이터(리뷰)의 고유 id**
  - `sentiment` : **영화 리뷰의 감성 레이블**
    - `1` : **긍정적 평가**
    - `0` : **부정적 평가**
  - `review` : **실제 영화평 텍스트**

#2-2 review 텍스트 예시 확인 및 전처리 필요성

- `review` **칼럼의 값을 하나 출력해 보면,** HTML **형식에서 추출되어** `<br />` **태그가 그대로 남아 있음.**

- `<br />` **같은** HTML **태그는 감성 분석을 위한 피처로 의미가 없으므로 제거하는 것이 좋음.**

EWHA EURON

#2 지도학습 기반 감성 분석 실습 – IMDB 영화평

```
import re

# <br /> 태그를 공백으로 변환
review_df['review'] = review_df['review'].str.replace('<br />', ' ', regex=False)

# 영어 알파벳이 아닌 문자를 공백으로 변환
review_df['review1'] = review_df['review'].apply(
    lambda x: re.sub('[^A-Za-z]', ' ', x)
)
```

#2-4 학습용 데이터/타깃 분리 및 train/test 분할

```
from sklearn.model_selection import train_test_split

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

# train/test split
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, 2), (7500, 2))
```

#2-3 문자열 처리: HTML 태그 및 불필요 문자 제거

2-3-1.
 태그 제거

:
를 공백 " "으로 바꾸기
감성 분석에 필요 없는 HTML 줄바꿈 태그 제거로 텍스트를 정리

2-3-2. 숫자/특수문자 제거

•숫자/특수문자 역시 감성 분석을 위한 핵심 피처가 아니라고 보고, 모두 공란으로 변경 → 정규 표현식 사용

•파이썬의 re 모듈을 사용해 정규식 기반 치환

•문자열 x에서 영어 대문자(A–Z)나 소문자(a–z)가 아닌 모든 문자를 찾아 " "(공백)으로 치환

→ 이렇게 하면 숫자, 구두점, 특수문자 등이 제거되어
영어 알파벳 + 공백 위주의 텍스트로 정리됨.

EWHA EURON

#2 지도학습 기반 감성 분석 실습 – IMDB 영화평

```python
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

# Pipeline 설정
pipeline = Pipeline([
    ('cnt_vect', CountVectorizer(stop_words='english', ngram_range=(1, 2))),
    ('lr_clf', LogisticRegression(solver='liblinear', C=10))
])

# 학습
pipeline.fit(X_train['review'], y_train)

# 예측
pred = pipeline.predict(X_test['review'])
pred_probs = pipeline.predict_proba(X_test['review'])[:, 1]

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

```
예측 정확도는 0.8868, ROC-AUC는 0.9501
```

#2-5 텍스트 벡터화 + ML 분류: Pipeline 활용

: 전처리된 리뷰 텍스트를 피처 벡터화하고,
그 벡터 기반 머신러닝 분류기를 학습해 성능 측정

2-5-1. Pipeline 구조

- CountVectorizer , TfidfVectorizer
  → **텍스트를 수치 벡터로 변환**
- LogisticRegression
  → **벡터를 입력으로 받아 감성을 이진 분류**

2-5-2. CountVectorizer 기반 성능 측정

- CountVectorizer**로 단어 빈도 기반 피처 벡터화**
- LogisticRegression **분류기에 입력해 학습**
- **평가 지표**:
  - **정확도**(Accuracy)
  - ROC-AUC

EWHA EURON

#2 지도학습 기반 감성 분석 실습 – IMDB 영화평

```
# 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(solver='liblinear', 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 {:.4f}, ROC-AUC는 {:.4f}'.format(
    accuracy_score(y_test, pred),
    roc_auc_score(y_test, pred_probs)
))
```

```
예측 정확도는 0.8935, ROC-AUC는 0.9597
```

#2-6 TF-IDF 벡터화 적용 및 성능 비교
: 같은 구조에서 벡터화 부분만 TfidfVectorizer로 변경
(Pipeline에서 `CountVectorizer` → `TfidfVectorizer`로 교체)

- 분류기: 여전히 LogisticRegression
- 평가 방법: 동일하게 정확도와 ROC-AUC 측정

2-6-1. 결과 비교
: TF-IDF 기반 피처 벡터화가 Count 기반보다 예측 성능이 조금 더 높게 나타남.

→ 즉, 단순한 단어 등장 횟수(Count)보다 단어의 상대적 중요도까지 고려하는 TF-IDF 방식이 감성 분석에 더 적합함.

#3 비지도학습 기반 감성 분석

#3-1 비지도학습 기반 감성 분석 개요

: 레이블(정답) 없는 텍스트 데이터에서 감성을 파악하기 위해 Lexicon(감성 사전)을 사용.

3-1-1. 지도 vs 비지도

- 지도 감성 분석은 데이터 세트에 `sentiment` 같은 레이블 값이 있음

→ 지도학습으로 분류 모델 학습.

- 하지만 실제 감성 분석용 데이터는 레이블이 없는 경우가 많음

→ 이럴 때 Lexicon 기반 비지도 분석이 유용.

* 언어 지원 문제

: 영어용 Lexicon은 다양하지만 한글을 지원하는 감성 사전은 거의 없음.

#3-2. Lexicon(감성 사전)의 개념

3-2-1 감성 사전의 구성

: 각 단어에 대해 긍정(Positive) 감성 또는 부정(Negative) 감성의 정도를 나타내는 수치(감성 지수)를 가짐.

3-2-2 감성 지수 결정 기준

: 단어 자체뿐 아니라 단어의 위치 / 주변 단어 / 문맥 / 품사(POS) 등을 참고해서 감성 지수가 정해짐.

3-2-3 NLTK의 감성 사전

: NLTK는 여러 서브 모듈을 가진 패키지이며, 그중에 Lexicon(감성 사전) 모듈도 포함. 이를 통해 감성 단어와 감성 점수 정보를 제공.

#3 비지도학습 기반 감성 분석

#3-3. WordNet : 시맨틱 기반 영어 어휘 사전

3-3-1. WordNet이란?

•NLTK에서 제공하는 WordNet 모듈은 방대한 영어 어휘 사전.
•단순한 "뜻 풀이" 수준의 사전이 아니라 시맨틱 분석 제공.

3-3-2. 시맨틱의 예시

: 같은 단어/문장이라도 문맥에 따라 의미가 달라짐

- 영어 단어 "present": "선물" / "현재"
- 우리말 "밥 먹었어?" : 정말로 식사 여부를 묻는 질문일 수도 있고, 단순한 안부 인사일 수도 있음.

3-3-3. Synset 개념

: WordNet은 시맨틱 정보 표현을 위해 Synset 사용

•특징

- 단순히 단어 하나가 아니라, 그 단어가 가지는 문맥·시맨틱 의미 묶음.
- 품사(명사, 동사, 형용사, 부사 등)별로 구성된 단어들을 Synset 단위로 표현.

#3-4. NLTK 대표 감성 사전들

3-4-1. SentiWordNet

- NLTK WordNet과 유사하지만 감성 단어 전용 WordNet.
- WordNet의 Synset 개념을 감성 분석에 특화시킨 구조.
- 각 Synset마다 3가지 감성 점수(긍정 감성 지수, 부정 감성 지수, 객관성 지수)를 가짐
  * 객관성 지수: 단어가 감성과 상관없이 얼마나 객관적인지를 나타냄.
- 문장 감성 계산
  1. 문장 안의 단어별 긍정 감성 지수와 부정 감성 지수를 합산해 최종 감성 지수를 계산.
  2. 그 값으로 문장이 긍정인지/부정인지 결정.

3-4-2. VADER

- 주로 소셜 미디어 텍스트에 대한 감성 분석용 패키지.
- 뛰어난 감성 분석 결과를 제공.
- 수행 속도가 빠르고, 대용량 텍스트 데이터에 적합.

3-4-3. Pattern

- 예측 성능 측면에서 가장 주목받는 패키지 중 하나.
- 단점 : 파이썬 3.x를 지원하지 않고, 파이썬 2.x에서만 동작(현재 기준).

#4 SentiWordNet 을 이용한 감성 분석

```python
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 개수 :', len(synsets))
print('synsets() 반환 값 :', synsets)
```

```
synsets() 반환 type : <class 'list'>
synsets() 반환 값 개수 : 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')]
```

```python
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 inclu
ding the moment of speech
Lemmas : ['present', 'nowadays']
##### Synset name : present.n.02 #####
POS : noun.possession
Definition : something presented as a gift
Lemmas : ['present']
```

#4-1. WordNet의 Synset 다루기

4-1-1. synsets()로 Synset 리스트 얻기

- Synset 객체의 표기 예:

`Synset('present.n.01')`

- `Present` : 단어
- `N` : 명사
- `01` : 해당 단어가 명사일 때 가지는 여러 의미 중 첫 번째 의미

4-1-2. Synset의 속성

- POS : 품사 정보
- Definition: 해당 Synset 의미에 대한 정의
- Lemma(부명제): Synset에 속하는 구체적인 단어들

4-1-3. 예시 Synset들의 의미

- `Synset('present.n.01')`
  - POS: `noun.time`
  - Definition: “시간적인 의미의 현재”
- `Synset('present.n.02')`
  - POS: `noun.possession`
  - Definition: “선물”

→ 같은 “present”라는 단어도 Synset에 따라 전혀 다른 의미로 표현되고, 이를 통해 WordNet이 시맨틱 정보를 구조적으로 표현함

EWHA EURON

#4 SentiWordNet 을 이용한 감성 분석

```python
# 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 synset을 반복하면서 다른 단어의 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 DataFrame 형태로 저장합니다.
similarity_df = pd.DataFrame(similarities, columns=entity_names, index=entity_names)
similarity_df
```

	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

#4-2 WordNet의 단어 간 유사도
: 어휘와 어휘 간 관계를 유사도로 표현 가능

- `path_similarity()` 메서드 제공:
  두 Synset 사이의 경로 기반 유사도를 계산.

•결과값
- `lion`과 `tree`의 유사도: 0.07 → 가장 낮음.
- `lion`과 `tiger`의 유사도: 0.33 → 가장 큼.

→ 어휘들 사이의 의미적 거리를 수치화할 수 있음.

#4 SentiWordNet 을 이용한 감성 분석

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

senti_synsets = list(swn.senti_synsets('slow'))

print('senti_synsets() 반환 type :', type(senti_synsets))
print('senti_synsets() 반환 값 개수 :', len(senti_synsets))
print('senti_synsets() 반환 값 :', senti_synsets)
```

```
senti_synsets() 반환 type : <class 'list'>
senti_synsets() 반환 값 개수 : 11
senti_synsets() 반환 값 : [SentiSynset('decelerate.v.01'), SentiSynset('slow.v.02'), SentiSynset
('slow.v.03'), SentiSynset('slow.a.01'), SentiSynset('slow.a.02'), SentiSynset('dense.s.04'), Se
ntiSynset('slow.a.04'), SentiSynset('boring.s.01'), SentiSynset('dull.s.08'), SentiSynset('slowl
y.r.01'), SentiSynset('behind.r.03')]
```

[13]:

```
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())
print('fabulous 객관성 지수:', fabulous.obj_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
fabulous 객관성 지수: 0.0
```

#4-3 SentiWordNet의 SentiSynset

4-3-1. SentiSynset의 지수

•감성 지수 : 긍정 감성 지수 / 부정 감성 지수
•객관성 지수 : 단어가 얼마나 감성과 관련 없이 객관적인지를 나타냄.
예) 어떤 단어가 전혀 감성적이지 않으면,

- 객관성 지수 = 1
- 긍정 감성 지수 = 0
- 부정 감성 지수 = 0

4-3-2. 예시: father vs fabulous

•Father(아버지)

- 객관성 지수: 1.0
- 긍정 감성 지수: 0
- 부정 감성 지수: 0

→ 감성 단어가 아니라 매우 객관적인 단어로 정의

•Fabulous(기막히게 멋진)

- 긍정 감성 지수: 0.875
- 부정 감성 지수: 0.125

→ 분명한 감성 단어, 긍정 쪽으로 강하게 치우친 단어로 해석.

#5 SentiWordNet을 이용한 영화 감상평 감성 분석

#5-1 SentiWordNet 감성 분석 전체 처리 순서

5-1-1 문서를 문장 단위로 분해

5-1-2 문장을 다시 단어 단위로 토큰화 + 품사 태깅

5-1-3 각 단어로부터 Synset → Senti_Synset 객체 생성

: 품사 태깅된 단어 + 품사 정보를 이용하여
- WordNet Synset 추출
- `senti_synsets()` 로 Senti_Synset 객체 생성

만약, synset이 존재하지 않으면 해당 단어는 감성 점수 계산에서 제외

5-1-4 각 단어의 감성 점수 추출 후 합산

: SentiSynset 객체는 세 가지 점수를 가짐
- 긍정 감성 지수 (pos_score)
- 부정 감성 지수 (neg_score)
- 객관성 지수 (objective_score)

분석에서는 긍정·부정 감성 점수만 활용
→ (긍정 점수 - 부정 점수) 또는 (긍정 + 부정) 등의 방식으로 합산

* 최종 판단:
- 총 감성 점수 ≥ 0 → 긍정(1)
- 총 감성 점수 < 0 → 부정(0)

→ 이 모든 과정을 자동으로 처리하기 위해
품사 태깅 함수, 감성 점수 계산 함수(swn_polarity)등을 별도로 구현

#5-2 품사 태깅 변환 함수

```python
# NLTK Penn Treebank POS Tag → WordNet POS Tag 변환 함수
def penn_to_wn(tag):
    if tag.startswith('J'):        # 형용사(Adjective)
        return wn.ADJ
    elif tag.startswith('N'):      # 명사(Noun)
        return wn.NOUN
    elif tag.startswith('R'):      # 부사(Adverb)
        return wn.ADV
    elif tag.startswith('V'):      # 동사(Verb)
        return wn.VERB
```

#5 SentiWordNet을 이용한 영화 감상평 감성 분석

```python
def swn_polarity(text):
    # 감성 지수 초기화
    sentiment = 0.0
    tokens_count = 0

    lemmatizer = WordNetLemmatizer()
    raw_sentences = sent_tokenize(text)

    # 문장 단위 처리
    for raw_sentence in raw_sentences:
        # 품사 태깅
        tagged_sentence = pos_tag(word_tokenize(raw_sentence))

        for word, tag in tagged_sentence:

            # WordNet 품사 변환
            wn_tag = penn_to_wn(tag)
            if wn_tag not in (wn.NOUN, wn.ADJ, wn.ADV, wn.VERB):
                continue

            # Lemmatization
            lemma = lemmatizer.lemmatize(word, pos=wn_tag)
            if not lemma:
                continue

            # Synset 생성
            synsets = wn.synsets(lemma, pos=wn_tag)
            if not synsets:
                continue

            # 첫 번째 synset 사용
            synset = synsets[0]

            # SentiSynset 생성
            try:
                swn_synset = swn.senti_synset(synset.name())
            except:
                continue

            # 감성 점수 = 긍정 - 부정
            sentiment += (swn_synset.pos_score() - swn_synset.neg_score())
            tokens_count += 1

    # 유효 단어가 없으면 중립(0) 처리
    if tokens_count == 0:
        return 0

    # 최종 감성 판단
    return 1 if sentiment >= 0 else 0
```

#5-3. swn_polarity(text) 함수 작성

swn_polarity(text) 함수 역할

- 문서 → 문장 단위로 분해
- 문장 → 단어 단위 토큰화
- 단어 → 품사 태깅
- 품사 기반 Synset/SentiSynset 생성
- 긍정/부정 점수 누적
- 점수 합산 후 감성 레이블 반환

* 최종 판단:

- 총 감성 점수 ≥ 0 → 긍정(1)
- 총 감성 점수 < 0 → 부정(0)

#5 SentiWordNet을 이용한 영화 감상평 감성 분석

```
review_df['preds'] = review_df['review'].apply(lambda x: swn_polarity(x))

y_target = review_df['sentiment'].values
preds = review_df['preds'].values
```

[18]:

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

# Confusion Matrix
print(confusion_matrix(y_target, preds))

print("정확도:", np.round(accuracy_score(y_target, preds), 4))
print("정밀도:", np.round(precision_score(y_target, preds), 4))
print("재현율:", np.round(recall_score(y_target, preds), 4))
```

```
[[ 4074  8426]
 [ 1345 11155]]
정확도: 0.6092
정밀도: 0.5697
재현율: 0.8924
```

#5-4. IMDB 감상평(review_df)에 적용

: 지도학습 기반 감성 분석에서 사용했던 `review_df` DataFrame 그대로 사용.
: 새로운 컬럼 `'preds` 추가

#5-5. SentiWordNet 감성 분석 결과 평가

•실제 정답 레이블: `review_df['sentiment']`
•예측 레이블: `review_df['preds']`

* 출력값
- 정확도(Accuracy)
- 정밀도(Precision)
- 재현율(Recall)

#5-6. SentiWordNet 감성 분석 성능 결과

- 정확도 Accuracy ≈ 61%
- 재현율 Recall ≈ 89%

→ 전반적인 성능은 만족스럽지 않음
→ 즉, SentiWordNet 기반 분석은 지도학습보다 성능이 떨어짐.
→ SentiWordNet은 WordNet 기반 시맨틱 감성 사전이라는 장점이 있지만 실전 감성 분석 성능이 낮음.

EWHA EURON

#6 VADER를 이용한 감성분석

#6-1. VADER를 이용한 감성 분석 개요

6-1-1. VADER Lexicon
: 소셜 미디어 텍스트(트위터, 댓글 등)의 감성 분석을 위해 만든 룰 기반 Lexicon.
: 감성 단어 리스트 + 규칙(대문자, 느낌표, 부정어, 이모지 등)을 활용해 감성 점수를 계산.

6-1-2 장점
: 구현이 간단하고, 속도가 빠르며,
특히 짧은 문장, 소셜 미디어에 특화된 감성 분석 성능이 좋음.

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

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

print(senti_scores)
```

```
{'neg': 0.131, 'neu': 0.742, 'pos': 0.127, 'compound': -0.8278}
```

#6-2 VADER

6-2-1. 분석기 객체 생성

6-2-2. 문장 하나에 대해 감성 점수 구하기
: `polarity_scores()` 는 딕셔너리 형태를 반환.

6-2-3. polarity_scores() 반환값 구조
`polarity_scores(text)` 의 키 값
- `'neg'` : 부정 감성 지수
- `'neu'` : 중립 감성 지수
- `'pos'` : 긍정 감성 지수
- `'compound'` : neg, neu, pos를 합성해 만든 종합 감성 점수
  - 값 범위: –1 ~ 1
    - –1에 가까울수록 매우 부정
    - +1에 가까울수록 매우 긍정

6-2-4. VADER로 문서별 감성 분류 기준
: 일반적으로 compound 점수를 기준으로 감성 판단
- compound ≥ 0.1 → 긍정 (1)
- compound < 0.1 → 부정 (0)

* 이때의 0.1이 임계값
: 상황에 따라 이 값을 높이거나 낮추면서 정밀도/재현율을 조절

EWHA EURON

#6 VADER를 이용한 감성분석

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

    # compound 값을 기준으로 threshold 이상이면 긍정(1), 아니면 부정(0)
    agg_score = scores['compound']
    final_sentiment = 1 if agg_score >= threshold else 0

    return final_sentiment


# review_df의 각 리뷰에 대해 vader_polarity 적용
review_df['vader_preds'] = review_df['review'].apply(lambda x: vader_polarity(x, 0.1))

# 실제 값과 예측 값 추출
y_target = review_df['sentiment'].values
vader_preds = review_df['vader_preds'].values

# 평가 지표 출력
print(confusion_matrix(y_target, vader_preds))
print("정확도:", np.round(accuracy_score(y_target, vader_preds), 4))
print("정밀도:", np.round(precision_score(y_target, vader_preds), 4))
print("재현율:", np.round(recall_score(y_target, vader_preds), 4))
```

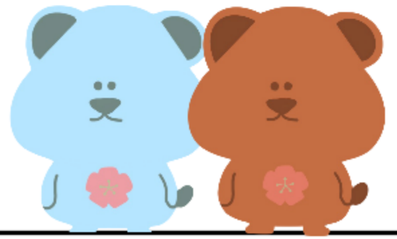
```
[[ 6842  5658]
 [ 1903 10597]]
정확도: 0.6976
정밀도: 0.6519
재현율: 0.8478
```

평가 지표	SentiWordNet	VADER
정확도	0.6092	0.6976
정밀도	0.5697	0.6519
재현율	0.7091	0.8478

- **정확도**(Accuracy): VADER가 0.6613 → 0.6956로 향상
- **정밀도**(Precision): 0.6472 → 0.6491로 상승
- **재현율**(Recall): 0.7091 → 0.8514로 매우 큰 폭으로 향상

→ VADER가 긍정/부정 중 특정 클래스(주로 긍정)를 더 잘 놓치지 않고 잡아냄

THANK YOU



EWHA EURON