



8장. 텍스트 분석

3팀 김남우 우정연 이덕주

목차

#01 텍스트 분석 개요

#02 텍스트 분석 이해

#03 텍스트 정규화

#04 Bag of Words - BOW

#05 감성 분석



8.0 텍스트 분석 개요



#8.0 텍스트 분석 개요

텍스트 분석이란?

비정형 데이터인 텍스트를 분석하는 작업

NLP vs. 텍스트 분석

◆ NLP

- 머신이 인간의 언어를 이해하고 해석하는 데 중점
- 언어를 해석하기 위한 기계 번역, 자동으로 질문을 해석하고 답을 해주는 질의응답 시스템 등의 영역
- 텍스트 분석을 향상하게 하는 기반 기술

◆ 텍스트 분석

- 비정형 텍스트에서 의미 있는 정보를 추출하는 것에 중점
- 머신러닝, 언어 이해, 통계 등을 활용해 모델을 수립하고 정부를 추출해 비즈니스 인텔리전스나 예측 분석 등의 분석 작업을 주로 수행

#8.0 텍스트 분석 개요

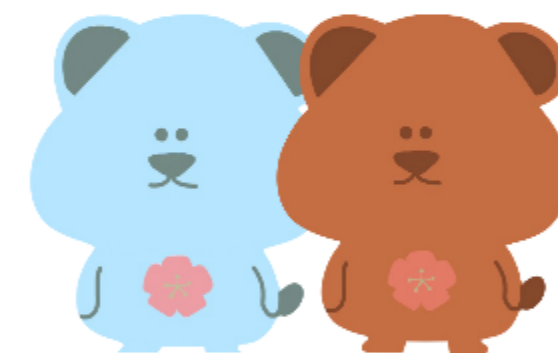
텍스트 분석의 종류

- 1. Descriptive analysis
 - 텍스트 집합에 있는 의미나 개념을 찾아내거나 이해를 돕는 형태
 - 문서 분류, 정보 검색, 단어 빈도 분석 등
- 2. Predictive analysis
 - 텍스트에 내포된 정보를 의사결정에 활용하는 형태
 - 과거 데이터를 검증하여 앞으로 벌어질 일들을 예측

텍스트 분석 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	

8.1 텍스트 분석 이해



#8.1 텍스트 분석 이해

머신러닝을 이용한 텍스트 분석

- 텍스트를 머신러닝에 적용하기 위해서는 비정형 텍스트 데이터를 어떻게 피처 형태로 추출하고 추출된 피처에 의미 있는 값을 부여하는가 하는 것이 매우 중요한 요소
- 피처 벡터화(Feature Vectorization) or 피처 추출(Feature Extraction)

피처 벡터화(Feature Vectorization)

- 텍스트를 word 기반의 다수의 피처로 추출하고 이 피처에 단어 빈도수와 같은 숫자 값을 부여하여 벡터값으로 표현
- e.g. BOW(Bag of Words), Word2Vec

#8.1 텍스트 분석 이해

텍스트 분석 수행 프로세스

0. 문제 정의: 해결하고자 하는 문제의 명확한 이해

1. 데이터 준비

- API 호출, 웹 크롤링 등

2. 데이터 전처리 ☆

- 데이터 정규화: 표현 방법이 다른 단어들 통합
- 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 개체로 인식
- 원형 복원: 형용사/동사 → 원형, 어간/어미 → 표현형으로 활용
- 불용어 제거: 조사, 접미사 등 제거
- 단어 빈도 분석: 불용어 및 빈출어의 제거 여부 & 필요한 단어들의 올바른 추출 확인

#8.1 텍스트 분석 이해

3. 데이터 분석

동시 출현 분석	문서 요약	텍스트 생성
키워드 추출	군집화	네트워크 분석
단어 임베딩	감성 분석	토픽 분석

4. 분석 결과 시각화

히스토그램	네트워크 다이어그램	덴드로그램
테이블	워드 클라우드	히트맵

5. 보고

- 분석 결과 비교/해석
- 분석 결과 보고

#8.1 텍스트 분석 이해

파이썬 텍스트 분석 패키지

◆ NLTK

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

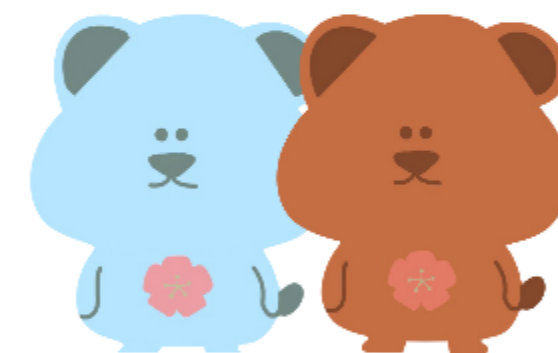
◆ SpaCy

- 뛰어난 수행 성능으로 최근 가장 주목을 받는 NLP 패키지
- 주요기능
 - 토큰화
 - POS 태깅
 - Sentence Boundary Detection (SBD)

◆ 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)

8.2 텍스트 정규화



#8.2 텍스트 정규화

텍스트 정규화

NLP 애플리케이션에 입력 데이터로 사용하기 위해
다양한 텍스트 데이터의 사전 작업을 수행하는 것을 의미



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

#8.2 클렌징&텍스트 토큰화

클렌징

텍스트에서 분석에 방해되는 불필요한 문자, 기호 등을 사전에 제거하는 작업
ex. HTML, XML 태그, 특정 기호 등

텍스트 토큰화

문서에서 문장을 분리하는 문장 토큰화와, 단어를 토큰으로 분리하는 단어 토큰화로 나눌 수 있음

#8.2 텍스트 토큰화

문장 토큰화(sent_tokenize)

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

```
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 your window or on 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!
```

#8.2 텍스트 토큰화

단어 토큰화(word_tokenize)

- 문장을 단어로 토큰화하는 것
- 기본적으로 공백, 콤마(,), 마침표(.), 개행문자(\n) 등으로 단어를 분리
- 정규 표현식을 이용하여 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', '.']
```

#8.2 텍스트 토큰화

단어 토큰화와 문서 토큰화 조합

```
from nltk import word_tokenize, sent_tokenize

def tokenize_text(text):

    ## 문장 토큰화
    sentences = sent_tokenize(text)
    ## 단어 토큰화
    word_tokens = [word_tokenize(sentence) for sentence in sentences]

    return word_tokens

## 문서 입력 및 문장/단어 토큰화 수행
word_tokens = tokenize_text(text_sample)

## 출력
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', 'go', 'to', 'work', ',', 'or', 'go', 'to', 'church', 'or', 'pay', 'your', 'taxes', '.']]
```


#8.2 텍스트 토큰화

단어 토큰화와 문서 토큰화 조합

문맥적 의미가 무시되는 문제 발생

» n_gram

- 연속된 n개의 단어를 하나의 토큰화 단위로 분리해 내는 것
- N개의 단어 크기 윈도우를 만들어 문장의 처음부터 오른쪽으로 움직이면서 토큰화 수행
- 예시: bigram(2_gram)
 - 연속적으로 2개의 단어들을 순차적으로 이동하며 단어들을 토큰화하는 과정
 - “Agent Smith knocks the door”
(Agent, Smith), (Smith, knocks), (knocks, the), (the, door)

#8.2 스톱 워드 제거

스톱워드

- 분석에 큰 의미가 없는 단어를 지칭
(ex. -is, a the, will 등 필수 문법 요소이지만 문맥적으로 큰 의미가 없는 단어)
- 이러한 단어는 사전에 제거하지 않으면 빈번함으로 인해 오히려 중요한 단어로 인지될 수 있음
- NLTK에서는 언어별 스톱 워드 목록을 제공

언어별 스톱워드 목록 다운로드

```
## NLTK에서 제공하는 언어별 스톱 워드 목록 다운로드
import nltk
nltk.download('stopwords')
```

스톱 워드 수 확인

```
## 영어 스톱 워드 수 확인
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']
```

#8.2 스톱 워드 제거

스톱 워드 제거

```
## 문장별로 단어를 토큰화해 생성된 word_tokens 리스트에 대해 불용어를 제거 후 분석을 위한 의미 있는 단어만 추출
stopwords = nltk.corpus.stopwords.words('english')
all_tokens = []

for sentence in word_tokens: # 3개의 각 문장을 확인
    filtered_words = []

    for word in sentence:

        # 대문자 -> 소문자
        word = word.lower()

        # 불용어가 아니라면, 리스트에 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', '.']]
```

#8.2 Stemming과 Lemmatization

- » 많은 언어에서 문법적인 요소에 따라 단어의 형태가 다양하게 변화
Stemming과 Lemmatization은 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

- Stemming에 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 Stemming과 Lemmatization

NLTK에서 Stemming 예시 (LancasterStemmer)

```
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

#8.2 Stemming과 Lemmatization

NLTK에서 Lemmatization 예시 (WordNetLemmatizer)

- 보다 정확한 원형 단어 추출을 위해 단어의 품사를 함께 입력해야 함 (ex. 동사는 v, 형용사는 a 등)
- Stemming 에 비해 더 정확하게 원형 단어를 추출해줄 것을 확인

```
import nltk
nltk.download('omw-1.4')

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
```

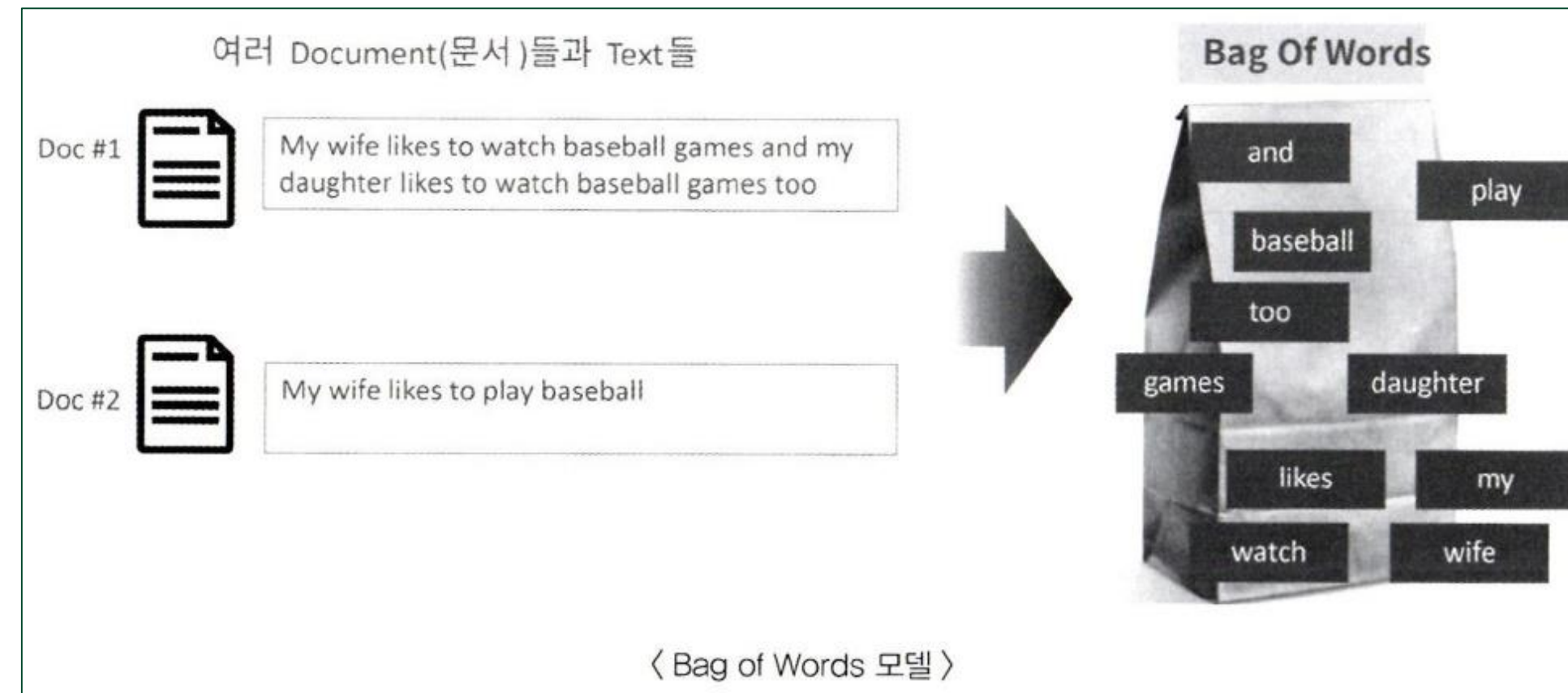
8.3 Bag of Words - BOW



#8.3 BOW 개념

- Bag of Words (BOW) 모델

- 문서가 가지는 모든 단어를 문맥이나 순서를 무시하고 일괄적으로 빈도 값을 부여해 피쳐 값을 추출하는 모델



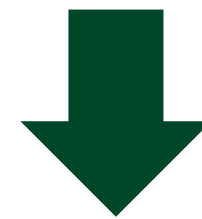
#8.3 BOW 개념

- 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”



Bag of Words의 단어 수 기반으로 피처 추출

1. 문장 1과 문장 2에 있는 모든 단어에서 중복을 제거하고 각 단어(feature 또는 term)를 칼럼 형태로 나열합니다. 그러고 나서 각 단어에 고유의 인덱스를 다음과 같이 부여합니다.

‘and’:0, ‘baseball’:1, ‘daughter’:2, ‘games’:3, ‘likes’:4, ‘my’:5, ‘play’:6, ‘to’:7, ‘too’:8, ‘watch’:9, ‘wife’:10

2. 개별 문장에서 해당 단어가 나타나는 횟수(Occurrence)를 각 단어(단어 인덱스)에 기재합니다. 예를 들어 baseball은 문장 1, 2에서 총 2번 나타나며, daughter는 문장 1에서만 1번 나타납니다.

	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

#8.3 BOW 개념

- **BOW 모델의 장점**

- 단순히 단어의 발생 횟수에 기반하여 쉽고 빠른 구축 가능
- 예상보다 문서의 특징을 잘 나타낼 수 있는 모델

- **BOW 모델의 단점**

- 단어의 순서를 고려하지 않으므로 문맥의 의미 반영 부족
 - > n_gram 기법 활용하여 보완 가능
- 대규모의 칼럼으로 구성된 행렬에서 대부분의 값이 0으로 채워지는 희소 행렬 문제
 - > 희소 행렬을 위한 특별한 기법이 마련되어 있음

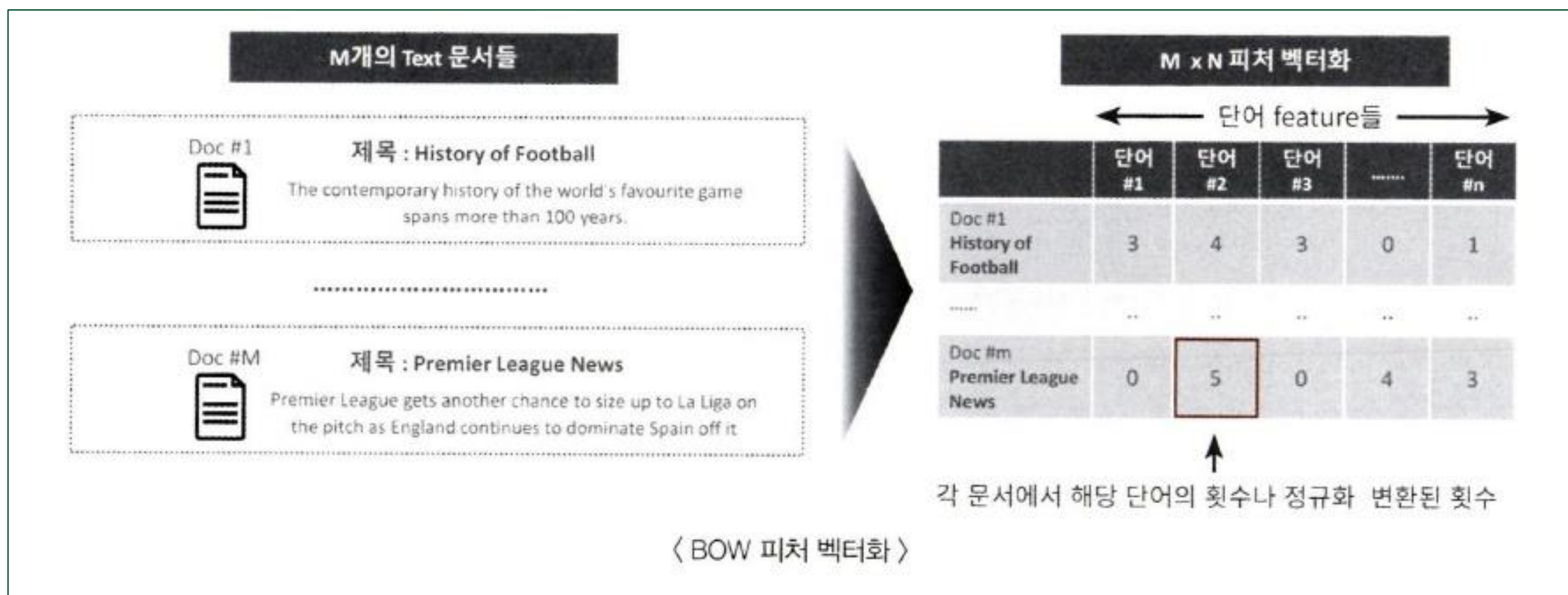
#8.3 BOW 피쳐 벡터화

- 피쳐 벡터화

- 텍스트를 특정 의미를 가지는 숫자형 값인 벡터 값으로 변환하는 것
- 각 문서의 텍스트를 단어로 추출해 피쳐로 할당하고, 각 단어의 발생 빈도와 같은 값을 피쳐에 값으로 부여해 각 문서를 이 단어 피쳐의 발생 빈도 값으로 구성된 벡터로 만드는 기법
- 기존 텍스트 데이터를 또 다른 형태의 피쳐 조합으로 변경 -> 넓은 범위의 피쳐 추출에 포함

- BOW 모델에서의 피쳐 벡터화

- 모든 문서에서 모든 단어를 칼럼 형태로 나열하고 각 문서에서 해당 단어의 횟수나 정규화된 빈도를 값으로 부여하는 데이터 세트 모델로 변경하는 것
- 카운트 벡터화/ TF-IDF





#8.3 BOW 피쳐 벡터화

- 카운트 벡터화

- 단어 피쳐에 값을 부여할 때 각 문서에서 해당 단어가 나타나는 횟수(Count)를 부여하는 경우
- 카운트 값이 높을수록 중요한 단어로 인식됨
- 언어의 특성상 자주 사용될 수 밖에 없는 단어까지 높은 값을 부여하게 되는 단점 있음

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

- 개별 문서에서 자주 나타나는 단어에 높은 가중치
- 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 패널티
- 텍스트가 길고 문서의 개수가 많은 경우 TF-IDF 방식이 더 좋은 예측 성능을 나타냄

 한 개의 문서(Document)	Term Frequency							
	The	Matrix	is	nothing	but	an	advertising	gimmick
	40	5	50	12	20	45	3	2
 모든 문서들(Corpus)	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

#8.3 사이킷런: CountVectorizer/TfidfVectorizer

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

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+\b'로, 공백 또는 개행 문자 등으로 구분된 단어 분리자(\b) 사이의 2문자문자 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.

#8.3 사이킷런: CountVectorizer/TfidfVectorizer



#8.3 BOW 벡터화를 위한 희소 행렬

- **희소 행렬**

- 대규모 칼럼으로 구성된 행렬에서 대부분의 값이 0으로 채워지는 행렬

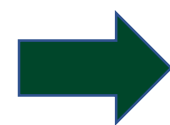
- **밀집 행렬**

- 대부분의 값이 0이 아닌 의미 있는 값으로 채워진 행렬

← 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	1
	문서
	문서 10000	0	1	3	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.

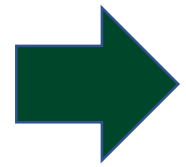


BOW 형태를 가진 언어 모델의 피쳐 벡터화는 대부분 희소 행렬임

#8.3 BOW 벡터화를 위한 희소 행렬

- **희소 행렬 단점**

- 메모리 공간이 많이 필요
- 행렬의 크기가 커서 연산 시, 시간이 많이 소요



희소 행렬을 물리적으로 적은 메모리 공간을 차지할 수 있도록 변환해야
대표적인 변환 방법 - COO 형식, CSR 형식

#8.3 희소 행렬 – COO 형식

- COO(Coordinate) 형식
 - 0이 아닌 데이터만 별도의 데이터 배열에 저장
 - 그 데이터가 가리키는 행과 열의 위치를 별도의 배열로 저장하는 방식
- 사이파이(Scipy)의 sparse 패키지
 - 희소 행렬 변환을 위한 다양한 모듈 제공

```
import numpy as np

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

# sparse 패키지의 coo_matrix를 이용해 COO 형식으로 희소 행렬 생성
sparse_coo = sparse.coo_matrix((data, (row_pos, col_pos)))

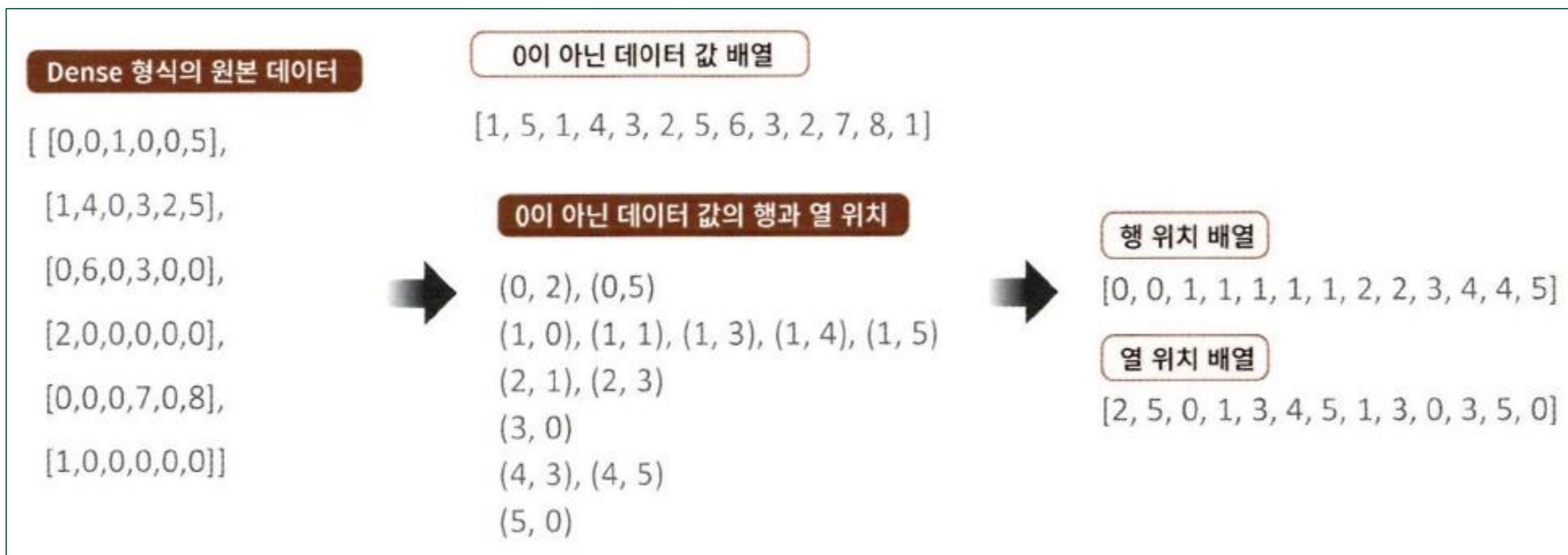
sparse_coo.toarray()

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

〈사이파이의 sparse를 이용한 COO 형식 수행〉

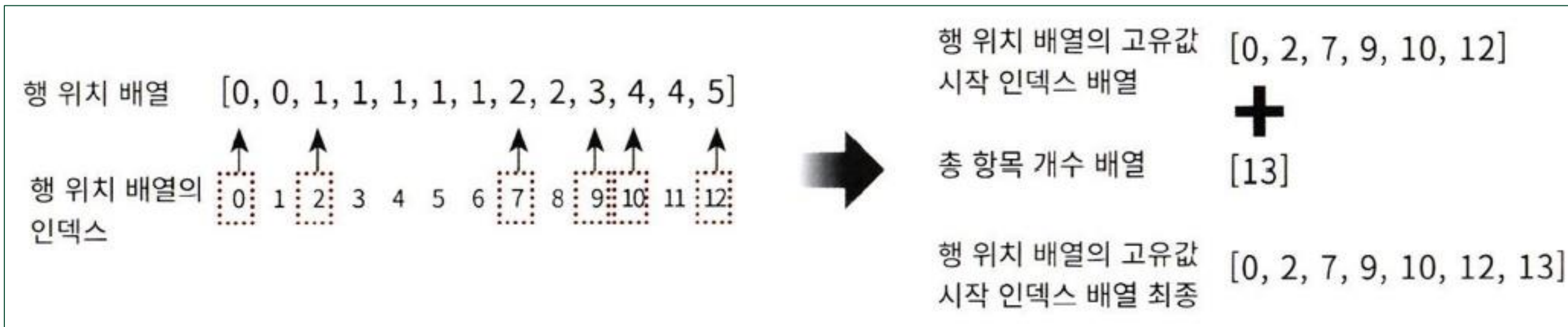
#8.3 희소 행렬 – CSR 형식

- CSR(Compressed Sparse Row) 형식
 - C00 형식이 행과 열의 위치를 나타내기 위해 반복적인 위치 데이터를 사용해야 하는 문제점을 해결한 방식
- C00 변환 형식의 문제점
 - 순차적인 같은 값이 반복적으로 나타남



EWCHA
EUROPEAN

행 위치 배열이 0부터 순차적으로 증가하는 값으로 이루어졌다는 특성 고려
 -> 행 위치 배열의 고유한 값의 시작 위치만 표기하는 방법으로 반복 제거 가능



- CSR

- 행 위치 배열 내에 있는 고유한 값의 시작 위치만 다시 별도의 위치 배열로 가지는 변환 방식 의미
- 고유 값의 시작 위치만 알고 있으면 얼마든지 행 위치 배열을 다시 만들 수 있음
- COO 방식보다 메모리가 적게 들고 빠른 연산 가능

#8.3 희소 행렬 – CSR 형식

〈사이파이의 csr_matrix 클래스를 이용한 CSR 변환〉

```
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 변환
sparse_csr = sparse.csr_matrix((data2, col_pos, row_pos_ind))

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

#8.3 희소 행렬 – CSR 형식

<실제 사용 시 -> 밀집 행렬을 생성 파라미터로 입력>

```
dense3 = 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]])  
  
coo = sparse.coo_matrix(dense3)  
csr = sparse.csr_matrix(dense3)
```

- 사이킷런의 CountVectorizer, TfidfVectorizer 클래스로 변환된 피처
- 벡터화 행렬은 모두 CSR 형태의 희소 행렬

8.5 감성 분석



#8.5 감성 분석

감성 분석(Sentiment Analysis)

- 문서의 주관적인 감성/의견/감정/기분 등을 파악하기 위한 방법
- 소셜 미디어, 여론조사, 온라인 리뷰, 피드백 등에서 활용
- 문서 내 텍스트가 나타내는 여러 가지 주관적인 단어와 문맥을 기반으로 감성(Sentiment) 수치를 계산
 - 감성 지수: 긍정 감성 지수 + 부정 감성 지수

지도학습	비지도학습
학습 데이터와 타깃 레이블 값을 기반으로 감성 분석 학습 수행 후 다른 데이터의 감성 분석 예측	‘Lexicon’ 이라는 감성 어휘 사전을 이용해 문서의 긍정적, 부정적 감성 여부 판단

#8.5 감성 분석

지도학습 기반 감성 분석 실습 – IMDB 영화평

데이터 로드

```
import pandas as pd
```

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

[Output]

	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...

Feature

- id: 각 데이터의 id
- sentiment: 영화평의 Sentiment 결과 값 / 1=긍정, 0=부정
- review: 영화평의 텍스트

텍스트 구성 확인 및 공백 변환

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

[Output]

```
"With all this stuff going down...<br />Visually ...<br />The actual feature film  
bit when it finally starts is only on for 20 minutes or so excluding the Smo...
```

- HTML 형식에서 추출해
 태그 존재
- 영어가 아닌 숫자/특수문자는 sentiment를 위한 피쳐로서 의미가 없음 → 공백으로 변경

```
import re
```

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

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


#8.5 감성 분석

Count 벡터화 및 ML 분류 알고리즘 적용

```
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 CountVectorization 수행.
# LogisticRegression의 C는 10으로 설정.
pipeline = Pipeline([
    ('cnt_vect', CountVectorizer(stop_words='english', ngram_range=(1, 2) )),
    ('lr_clf', LogisticRegression(C=10))])

# Pipeline 객체를 이용해 fit(), predict()로 학습/예측 수행. predict_proba()는 roc_auc 때문에 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)))
```

【Output】

예측 정확도는 0.8860, ROC-AUC는 0.9503

TF-IDF 벡터화 및 ML 분류 알고리즘 적용

```
# 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)))
```

【Output】

예측 정확도는 0.8936, ROC-AUC는 0.9598

#8.5 감성 분석

비지도학습 기반 감성 분석

- Lexicon 기반 (레이블 값 x)

Lexicon

- 감성 어휘 사전
- 긍정(Positive) 감성 or 부정(Negative) 감성의 정도를 의미하는 수치 → 감성 지수(Polarity score)
- 감성 지수는 단어의 위치, 주변 단어, 문맥, POS 등을 참고해 결정
- NLTK 패키지의 모듈로 포함되어 있음

WordNet

- NLP 패키지 속 모듈로 방대한 영어 어휘 사전
- 시맨틱(문맥상 의미) 분석 제공
- 각각의 품사로 구성된 개별 단어를 WordNet의 핵심 개념인 Synet(Sets of cognitive synonyms)을 이용해 표현
- 예측 성능이 좋지 못함

SentiWordNet	감성 단어 전용의 WordNet 구현
VADER	소셜 미디어의 텍스트에 대한 감성 분석 제공을 위한 패키지
Pattern	예측 성능이 가장 뛰어난 패키지

#8.5 감성 분석

SentiWordNet을 이용한 감성 분석

NLTK 데이터 세트 및 패키지 다운로드

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

Synset 객체 반환

```
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)
```

[Output]

```
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')]
```

Synset 속성

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

[Output]

```
#### 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']
```


#8.5 감성 분석

단어 상호 유사도

- path_similarity() 메서드

```
# 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

← lion은 tree와의 유사도가 0.07로 가장 적고,
tiger와는 유사도가 0.33으로 가장 큼.

SentiWordNet – Senti_Synset

- senti_synsets()

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

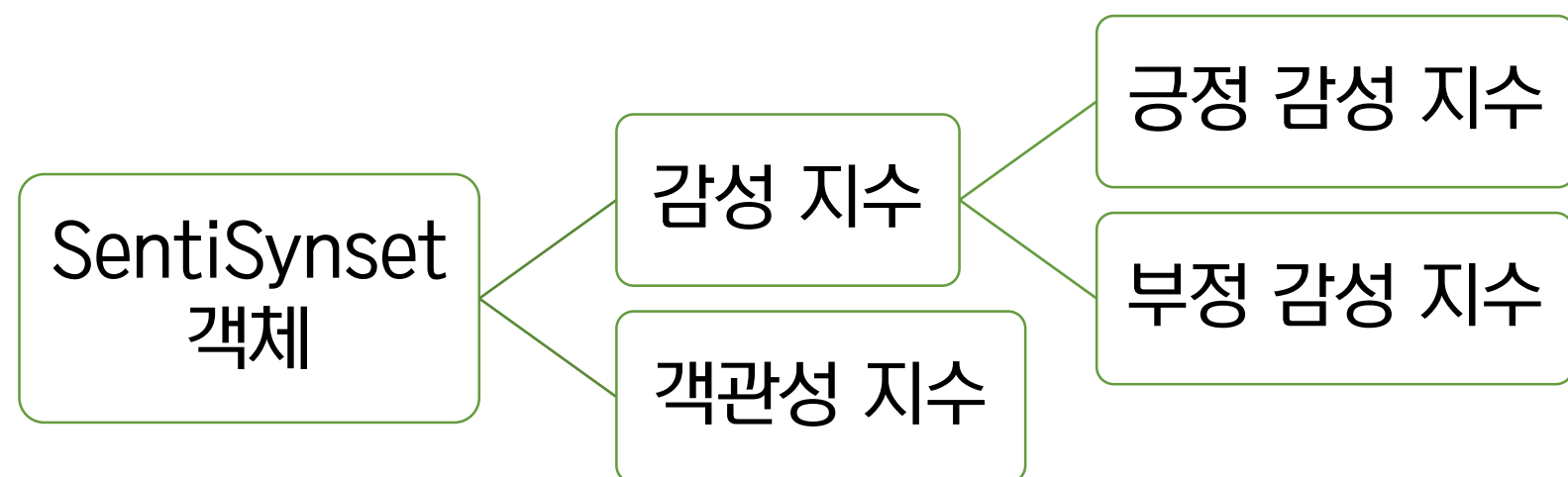
senti_synsets = list(swn.senti_synsets('slow'))
print('senti_synsets() 반환 type :', type(senti_synsets))
print('senti_synsets() 반환 값 개수:', len(senti_synsets))
print('senti_synsets() 반환 값 :', senti_synsets)
```

【Output】

```
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'), SentiSynset('slow.a.04'), SentiSynset('boring.s.01'), SentiSynset('dull.s.08'),
SentiSynset('slowly.r.01'), SentiSynset('behind.r.03')]
```

#8.5 감성 분석

감성 지수와 객관성 지수



```
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())
```

[Output]

```
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
```

#8.5 감성 분석

SentiWordNet을 이용한 영화 감상평 감성 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

품사 태깅 수행 함수 생성

- SentiWordNet을 이용하기 위해 WordNet을 이용해 문서를 다시 단어로 토큰화한 뒤 어근 추출과 품사 태깅을 적용해야 함

```
from nltk.corpus import wordnet as wn

# 간단한 NLTK PennTreebank Tag를 기반으로 WordNet기반의 품사 Tag로 변환
def penn_to_wn(tag):
    if tag.startswith('J'):
        return wn.ADJ
    elif tag.startswith('N'):
        return wn.NOUN
    elif tag.startswith('R'):
        return wn.ADV
    elif tag.startswith('V'):
        return wn.VERB
```


#8.5 감성 분석

Polarity Score 합산 함수 생성

- 문장 → 단어 토큰 → 품사 태깅 후 SentiSynset 클래스 생성하고 Polarity Score 합산
- 총 감성 지수(긍정 감성 지수 + 부정 감성 지수)가 0 이상일 경우 긍정 감성, 미만이라면 부정 감성으로 예측

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import sentiwordnet as swn
from nltk import sent_tokenize, word_tokenize, pos_tag

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

    lemmatizer = WordNetLemmatizer()
    raw_sentences = sent_tokenize(text)
    # 분해된 문장별로 단어 토큰 → 품사 태깅 후에 SentiSynset 생성 → 감성 지수 합산
    for raw_sentence in raw_sentences:
        # NLTK 기반의 품사 태깅 문장 추출
        tagged_sentence = pos_tag(word_tokenize(raw_sentence))
        for word, tag in tagged_sentence:

            # WordNet 기반 품사 태깅과 어근 추출
            wn_tag = penn_to_wn(tag)
            if wn_tag not in (wn.NOUN, wn.ADJ, wn.ADV):
                continue
            lemma = lemmatizer.lemmatize(word, pos=wn_tag)
            if not lemma:
                continue
```

```
# 어근을 추출한 단어와 WordNet 기반 품사 태깅을 입력해 Synset 객체를 생성.
synsets = wn.synsets(lemma, pos=wn_tag)
if not synsets:
    continue
# sentiwordnet의 감성 단어 분석으로 감성 synset 추출
# 모든 단어에 대해 긍정 감성 지수는 +로 부정 감성 지수는 -로 합산해 감성 지수 계산.
synset = synsets[0]
swn_synset = swn.senti_synset(synset.name())
sentiment += (swn_synset.pos_score() - swn_synset.neg_score())
tokens_count += 1

if not tokens_count:
    return 0

# 총 score가 0 이상일 경우 긍정(Positive) 1, 그렇지 않을 경우 부정(Negative) 0 반환
if sentiment >= 0 :
    return 1

return 0
```

#8.5 감성 분석

긍정 및 부정 감성 예측

```
train_df['preds'] = train_df['review'].apply( lambda x : swn_polarity(x) )
y_target = train_df['sentiment'].values
preds = train_df['preds'].values
```

SentiWordNet의 감성 분석 예측 성능

```
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
```

```
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))
```

【Output】

```
[[7668 4832]
 [3636 8864]]
정확도: 0.6613
정밀도: 0.6472
재현율: 0.7091
```


#8.5 감성 분석

VADER를 이용한 감성 분석

VADER

- 소셜 미디어의 감성 분석 용도로 만들어진 룰 기반의 Lexicon
- SentimentIntensityAnalyzer 클래스로 쉽게 감성 분석 가능

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

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

【Output】

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

#8.5 감성 분석

VADER를 이용한 IMDB 감성 분석

Vader_polarity() 함수 생성

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

    # compound 값에 기반해 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 이용해 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(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))
```

【Output】

```
[[ 6736  5764]
 [ 1867 10633]]
정확도: 0.6948
정밀도: 0.6485
재현율: 0.8506
```

- 정확도가 SentiWordNet보다 향상됨

※ 지도학습 분류 기반의 예측 성능에 비해 낮은 수준이지만 결정 클래스 값이 없는 상황에서 일정 수준 만족할 만한 수치가 도출됨

THANK YOU

